X.25
PROGRAMMER'S MANUAL

IDACOM

A division of
HEWLETT PACKARD

X.25 PROGRAMMER'S MANUAL

September 1990
Version 2.0

X.25 MONITOR

Version 2.0

This release notice contains any changes specific to this application only. Refer to the release notices in the Programmer's Reference Manual update for a list of Home processor and application independent changes.

1.1 Enhancements

- **Fast Capture**
  Fast capture to RAM or disk is used if the application is not using any of the following features:
  - Test Manager
  - Filters
  - Triggers
  - Throughput Graph

- **Protocol Standard**
  The protocol standard (1980/1984 CCITT Recommendations) can be selected for decoding.

- **Frame and Packet Decoding**
  All frames and packets are decoded correctly for the selected protocol standard including individual fields in each packet.

  Example:
  Reset cause code hex 1D must be decoded as 'NETWORK OUT OF ORDER' for X.25 (1984), and 'INVALID' for X.25 (1980).

- **Facility Field Decoding**
  All facilities are decoded correctly for the selected standard.

- **Registration Packet Decoding**
  Registration packets are decoded if the selected standard is X.25 (1984).

1.2 Problems Fixed

- **Invalid GFI**
  An invalid GFI is now decoded correctly in complete report format.

- **Selective Address Filters**
  Clear confirm packets with the selected address are now displayed, captured, or recorded.

X.25 EMULATION

Version 2.0

2.1 Enhancements

- **Emulation Configuration**
  The Emulation Configuration Menu has been split into the Setup Menu (physical layer), the Frame Layer Menu, and the DCE/DTE Packet Layer Menu.

  The emulation can be configured to emulate a DTE which conforms to either X.25 1980/1984 CCITT Recommendations.

  Modulo detection (sequence numbering) can be manual or automatic.

- **Facilities**
  The Facility Menu has been added to define call packet facilities and call and clear user data.

- **Send Topics**
  The **Send** topic has been split into the **L2Send** and **L3Send** topics allowing more frames and packets to be sent via function keys.

- **Busy Condition**
  The layer 2 busy condition can now be set manually under the **Emulation** topic.

- **Q–bit Checking**
  Data packets are now checked for the Q–bit.

- **Interrupt User Data**
  The maximum support length of the interrupt user data is 32 octets for X.25 (1984) and 1 octet for X.25 (1980).

2.2 Problems Fixed

- **Effect of Link Level on Packet Level**
  When layer 2 enters the disconnected state, layer 3 sets P(S) and P(R) to 0.

- **Reset the Link**
  When the link is reset, an SABM or SABME is transmitted for modulo 8 or 128 respectively.

- **Invalid Frame Decoding**
  The P/F variable is only set if the received frame is valid.

DM Response
The emulation now sends a SABM only if the received DM/F=0.

Long I Frame
The emulation now retransmits FRMR when a long I frame with P=1 is received.

Invalid Response Frame
All invalid response frames are now discarded; the emulation only responds to invalid command frames.

Decoding
The diagnostic field is now decoded correctly according to the selected emulation mode. The diagnostic field is mandatory for clear, restart, and reset request packets, and optional for clear, restart, and reset indications.

Diagnostic Codes
The emulation now uses correct diagnostic codes:

- Long clear confirmation packet in p2 DTE waiting state is 21 instead of 39.
- Long incoming call packet in p5 call collision state is 39 instead of 24.
- Long clear confirmation packet in p5 call collision state is 24 instead of 39.

Emulation State
When a call connected packet is received while in p5 call collision state, the DTE will now accept the packet and the state changes to p4 data transfer.

PREFACE

This manual is intended to provide a programmer's guide to the X.25 Monitor/Emulation programs. General programming information is provided in the Programmer's Reference Manual. Information contained in this manual is machine independent.

This manual is not intended to provide basic user instruction, but rather addresses the issues of writing test programs using the Interactive Test Language (ITL). Refer to the machine specific User Manual for a quick reference to the basic operation of the protocol tester.

P/N 6000-1200

IDACOM Electronics Ltd.
A division of Hewlett-Packard

4211 - 95 Street
Edmonton, Alberta
Canada T6E 5R6
Phone: (403) 462-4545
Fax: (403) 462-4869

TABLE OF CONTENTS

TABLE OF CONTENTS [continued]

TABLE OF CONTENTS [continued]

TABLE OF CONTENTS [continued]

LIST OF FIGURES

LIST OF TABLES

1
INTRODUCTION

The X.25 Monitor is implemented in accordance with the CCITT X.25 (1980/1984) Recommendations for single link or multilink procedures. It is not a state–driven monitor, i.e. it does not have the knowledge of expected events. Rather, the monitor decodes and reports information on received frames and lead changes. Filters, triggers, RAM capture, and disk recording are also available.

The X.25 Emulation is implemented as a multi–layer, state–driven protocol emulation together with an integral protocol monitor. Each protocol layer has separate program modules that communicate with each other to implement protocol operation. The emulation has been set up to run as:

- an automatic simulation which operates precisely in accordance with the CCITT X.25 (1980/1984) Recommendations for single link procedure and modulo 8 or modulo 128;
- a semi–automatic tester. The test manager is used to build and execute test scenarios for generation of errors and to test responses to all protocol layers. Multilink procedure can be tested under test scenario control; and
- a manual tester. The test is controlled from the user's keyboard.

All user test scripts are written in the ITL language. Test programs are made up of sequences of ITL commands (or 'words') which exchange data and parameters via a Last In First Out (LIFO) stack. All commands consume zero or more parameters from the stack (input) and/or leave results on the stack (output). These commands have a stack effect comment shown beside the definition of the command to define its input and output parameters.

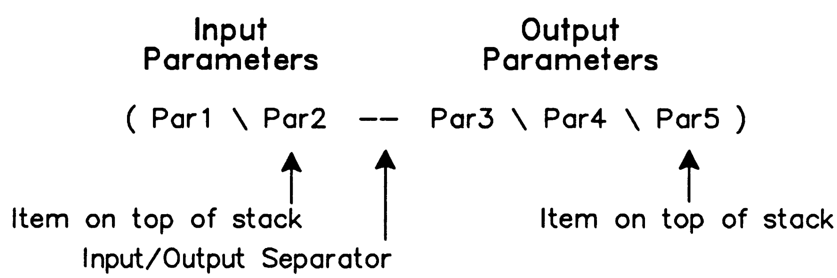


Figure 1–1 Sample Stack Comment

NOTE
See Appendix C for further explanation of stack parameters.

Sample complete test scripts are supplied in Section 14. These test scripts are also supplied on disk with the application program.

The X.25 applications can be controlled remotely from a terminal. All commands described in this manual can be entered from the remote terminal's keyboard followed by a ↵ (RETURN). The application processes the remote command and returns the 'ROK' prompt to the remote terminal. The remote terminal must be connected to the modem port on the back of the tester. To configure the application for remote control, refer to the Programmer's Reference Manual.

2
MONITOR CONFIGURATION

This section describes the commands associated with each item on the Monitor Configuration Menu.

```
                Monitor Configuration Menu

   Protocol Standard                 X.25(1984)
 → Interface Type                    RS232C/V.28
   Interface Leads                   DISABLED
   Bit Rate                          UNKNOWN
   Modulo Detection                  AUTOMATIC
   Link Access Procedure             LAPB
   Frame Sequence Number Modulo      MOD 8
   Link Procedure                    SINGLE LINK
```

Figure 2–1 Monitor Configuration Menu

WAKEUP_CPU (--)
Initializes the X.25 protocol for the monitor and configures the physical interface.

NOTE
Use WAKEUP_CPU once after all physical changes are made.

→ *Protocol Standard*
Selects protocol standard for monitor decoding.

STD=NONE (--)
Decodes a received frame according to the user setting..

NONE function key

STD=X25(80) (--)
Decodes a received frame according to the CCITT X.25 (1980) Recommendation.

X.25 (1980) function key

STD=X25(84) (--)
Decodes a received frame according to the CCITT X.25 (1984) Recommendation (default).

X.25 (1984) function key

The following table shows the effects of Protocol Standard on the monitor decoding.

Parameters/Procedures	NONE	X.25(1980)	X.25(1984)
Link Procedure	Single Link or Multilink	Single Link	Single Link or Multilink
Facility Field Decoding:			
Called Line Address Modified Notification	Supported	Not Supported	Supported
Closed User Group Extended Format	Supported	Not Supported	Supported
Closed User Group with Outgoing Access Basic and Extended Format	Supported	Not Supported	Supported
Transit Delay	Supported	Not Supported	Supported
Charging Information	Supported	Not Supported	Supported
Call Duration	Supported	Not Supported	Supported
Segment Count	Supported	Not Supported	Supported
Monetary Unit	Supported	Not Supported	Supported
Call Redirection Notification	Supported	Not Supported	Supported
RPOA Extended Format	Supported	Not Supported	Supported
Network User Identification	Supported	Not Supported	Supported
Registration Packet Decoding	Supported	Not Supported	Supported

Table 2–1 Monitor Protocol Standard Parameters/Procedures

→ *Interface Type*

IF=V28 (--)

Selects the V.28/RS-232C connector (default) and electrically isolates the other connectors on the port.

RS232C/V.28 function key

IF=V11 (--)

Selects the V.11/X.21 connector and electrically isolates the other connectors on the port.

RS422/V.11 function key

IF=V35 (--)

Selects the V.35 connector and electrically isolates the other connectors on the port.

V.35 function key

IF=V36 (--)

Selects the V.36 connector and electrically isolates the other connectors on the port.

RS449/V.36 function key

NOTE
A WAN tester has a V.28, V.11, and either a V.35 or V.36 connector. These commands are only applicable if the program is running on a WAN interface.

→ *Interface Leads*
Individual or all interface leads can be enabled or disabled. Leads must be enabled for test manager detection.

ENABLE_LEAD (lead identifier --)
Enables the specified lead. Refer to the Programmer's Reference Manual for a list of supported leads for each interface type.

Example:
Enable the request to send lead.
```
IRS ENABLE_LEAD
```

DISABLE_LEAD (lead identifier --)
Disables (default) the specified lead. Refer to the Programmer's Reference Manual for a list of supported leads for each interface type.

Example:
Disable the clear to send lead.
```
ICS DISABLE_LEAD
```

ALL_LEADS (-- lead identifier)
Enables/disables all leads supported on the currently selected WAN interface. ALL_LEADS must be used with ENABLE_LEAD or DISABLE_LEAD.

Example 1:
Enable all leads on the current interface.
```
ALL_LEADS ENABLE_LEAD
```

ENABLED function key

Example 2:
Disable all leads on the current interface.
```
ALL_LEADS DISABLE_LEAD
```

DISABLED function key

→ *Bit Rate*
The interface speed is measured, in bits per second, directly from the physical line.

INTERFACE-SPEED (-- address)
Contains the current bit rate (default value is 64000) and is used by the monitor to calculate throughput measurements.

→ *Modulo Detection*

AUTO-MOD (-- address)

Contains the current setting of the modulo (sequence numbering) detection mechanism when a SABM or SABME is received. Valid values are 1 for automatic (default) or 0 for manual.

NOTE

When a SABM or SABME is received, the program is automatically placed into modulo 8 or 128, respectively (only if automatic modulo detection is selected).

→ *Link Access Procedure*

L2=LAP (--)

Decodes frames according to LAP procedure (i.e. SARM and CMDR).

LAP function key

L2=LAPB (--)

Decodes frames according to LAPB (default) procedure (i.e. DM and FRMR).

LAPB function key

→ *Frame Sequence Number Modulo*

L2=MOD8 (--)

Selects modulo 8 method of decoding (default).

MOD 8 function key

L2=MOD128 (--)

Selects modulo 128 method of decoding.

MOD 128 function key

→ *Link Procedure*

SLP (--)

Uses single link procedure for decoding and reporting (default). Frames having addresses of 0x01 and 0x03 are decoded.

SINGLE LINK function key

MLP (--)

Uses multilink procedure for decoding and reporting. This mode displays the control bits and sequence numbers present in the multilink header. Frames having addresses of 0x07 and 0x0F are decoded.

MULTILINK function key

The following variables are set with the MLP and SLP commands. The contents of these variables can be obtained with the @ (fetch) operation.

ADDRESS–A (-- address)
Contains the expected X.25 frame address for commands from the DCE or responses from the DTE. The default value is 3 for single link procedure, and 15 for multilink procedure.

NOTE
In the CCITT X.25 (1984) Recommendation, this address is referred to as Address C for MLP. MLP can only be selected if CCITT X.25 (1984) Recommendation is selected as the protocol standard.

ADDRESS–B (-- address)
Contains the expected X.25 frame address for commands from the DTE or responses from DCE. The default value is 1 for single link procedure, and 7 for multilink procedure.

NOTE
In the CCITT X.25 (1984) Recommendation, this address is referred to as Address D for MLP. MLP can only be selected if CCITT X.25 (1984) Recommendation is selected as the protocol standard.

3
MONITOR ARCHITECTURE

The X.25 Monitor program monitors live data, saves data to capture RAM or disk, and displays data in a number of different formats. Data can be passed through filters which limit the displayed, captured, or recorded data. Triggers perform specific actions when a specified event occurs.

3.1 Live Data

The monitor application receives events from the interface or from the internal timer and processes them as shown in Figure 3–1.

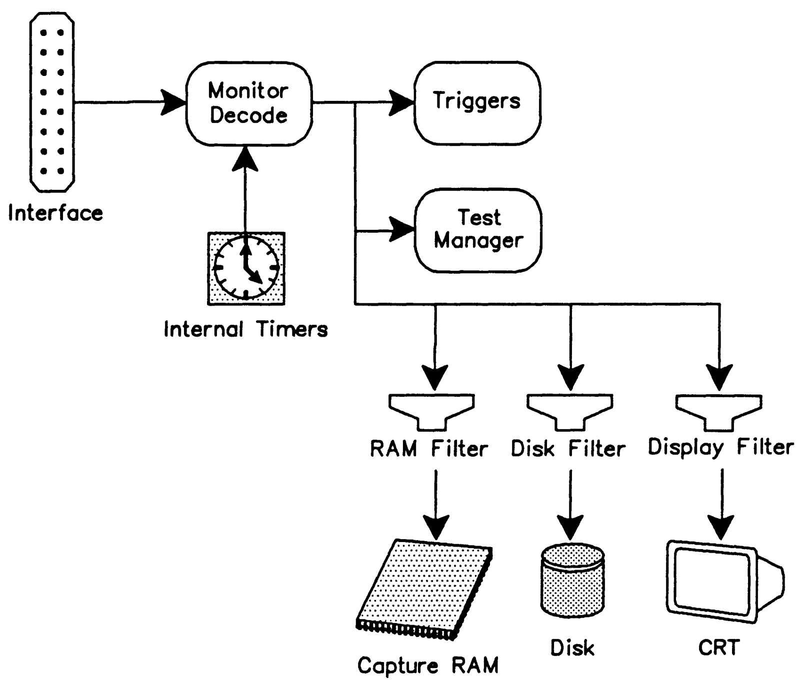


Figure 3–1 X.25 Monitor Data Flow Diagram – Live Data

By default, the X.25 Monitor captures data in the capture RAM buffer and displays it on the screen in a short format report.

Display topic
Live Data function key

MONITOR (––)
Selects the live data mode of operation. All incoming events are decoded and displayed in real–time.

3.2 Playback

Data (both protocol and lead information) can be examined in an offline mode using either the capture RAM or the disk file as the data source.

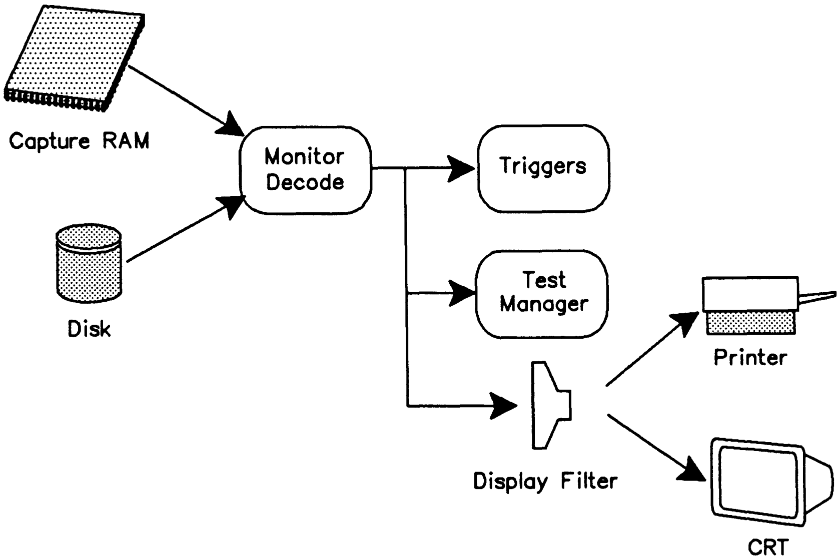


Figure 3–2 X.25 Monitor Data Flow Diagram – Offline Processing

FROM_CAPT HALT
Display topic
Playback RAM function key

FROM_DISK HALT PLAYBACK
Display topic
Playback Disk function key

HALT (––)
Selects the playback mode of operation. Data is retrieved from capture RAM or a disk file, decoded, and displayed or printed. Capture to RAM is suspended in this mode.

FROM_CAPT (––)
Selects the capture buffer as the source for data transfer.

FROM_DISK (––)
Selects a disk file as the source for data transfer.

PLAYBACK (--)
Opens a data recording file for playback. When used in the Command Window, the filename can be specified as part of the command.

Example:
```
PLAYBACK DATA1
```

NOTE
When PLAYBACK is used in a test script, the filename must be specified with =TITLE.

=TITLE (filename --)
Specifies the name of the file to open for disk recording or disk playback.

Example:
Obtain playback data from disk.

```
FROM_DISK          ( Identify a disk file as data source )
HALT               ( Place the monitor in playback mode )
" X25DAT" =TITLE   ( Create title for next data file to be opened )
PLAYBACK           ( Playback data )
```

Playback Control

The following commands control display scrolling.

FORWARD or **F** (--)
Scrolls one line forward on the screen.

⇓ (Down arrow)

BACKWARD or **B** (--)
Scrolls one line backward on the screen.

⇑ (Up arrow)

SCRN_FWD or **FF** (--)
Scrolls one page forward on the screen.

CTRL ⇓

SCRN_BACK or **BB** (--)
Scrolls one page backward on the screen.

CTRL ⇑

TOP (--)
Positions the display at the beginning of the playback source.

CTRL SHIFT ⇑

BOTTOM (--)
Positions the display at the end of the playback source.

CTRL SHIFT ⇓

3.3 Simultaneous Live Data and Playback

Live data can be recorded to disk while playing back data from capture RAM.

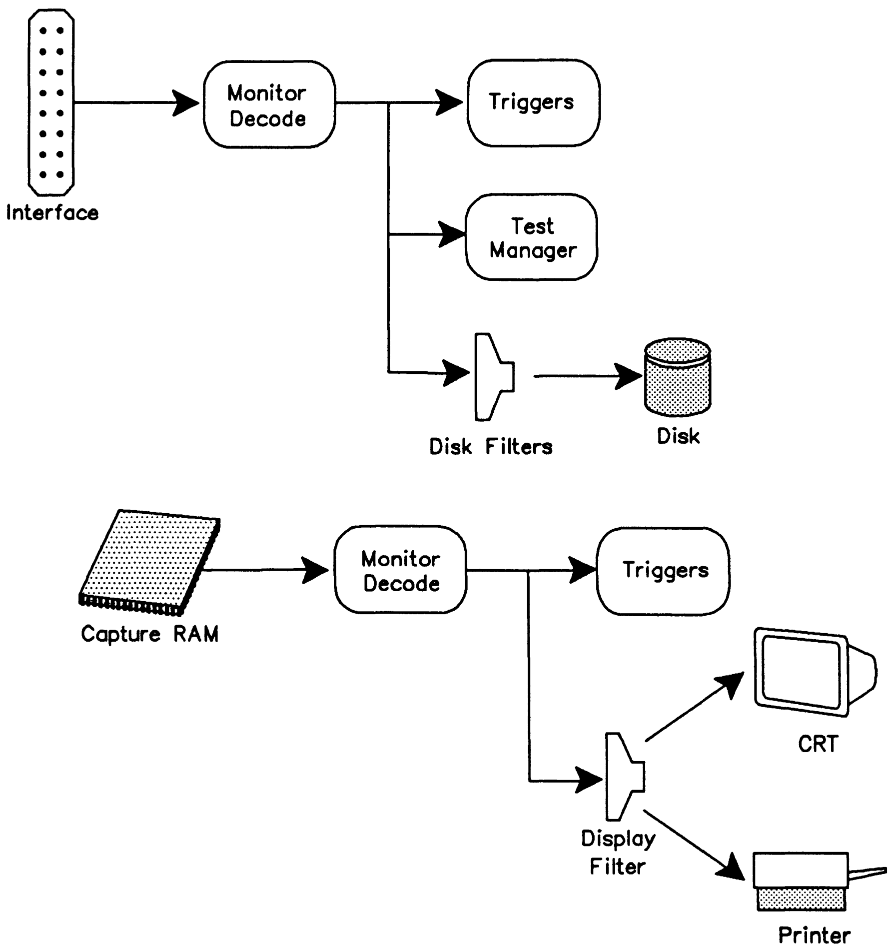


Figure 3–3 X.25 Monitor Data Flow Diagram – Freeze Mode

FROM_CAPT FREEZE
Capture topic
Record to DISK function key
Display topic
Playback RAM function key

FREEZE (--)
Enables data to be recorded to disk while data from capture RAM is played back.

4 MONITOR DECODE

The X.25 Monitor supports both SLP (single link procedure) and MLP (multilink procedure) decoding and display.

Data or lead changes from the interface, capture RAM, or disk file are decoded by protocol layer. Each decoding layer stores information in a pool of variables for later use by either a test program or other parts of the monitor application.

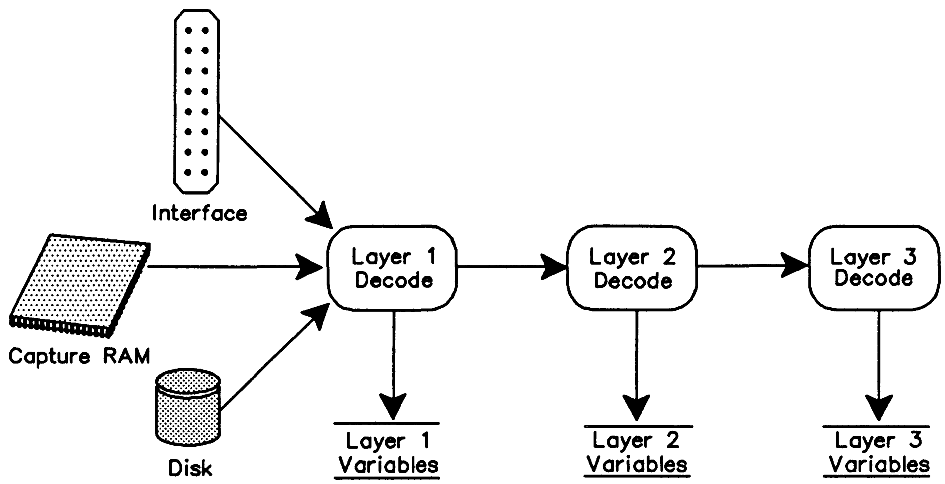


Figure 4–1 X.25 Monitor Data Flow Diagram – Decode

4.1 Communication Variables

The following variables are set during the decode process and contain protocol specific information as defined in the CCITT X.25 (1980 or 1984) Recommendation.

NOTE
These variables can be read using the @ (fetch) operation.

Layer 1

The layer 1 decode operation saves information concerning frame length, timestamps, port identifier, and block sequence number. For lead transitions, information is saved concerning the changed lead(s) and; for timers, the number of the expired timer.

PORT–ID (-- address)
Contains a 2 byte value identifying the received direction for data. The lower byte indicates the TO_DCE (hex value 08) or TO_DTE (hex value 20) receive stream. The upper byte indicates the application processor that received the frame.

Example:
Determine the direction of the received stream.

```
PORT-ID @
OXFF AND        ( The AND operation eliminates the upper byte )
```

This operation leaves the received stream direction on the stack. It is 0 for a trace statement, or equal to one of the following predefined constants: TO_DTE_RX for data to the terminal or TO_DCE_RX for data to the network. For further explanation of port identification, consult the Programmer's Reference Manual.

START–TIME (-- address)
Contains the 48 bit start of frame timestamp for data. Use with the GET_TSTAMP_MILLI or GET_TSTAMP_MICRO commands. See the Programmer's Reference Manual.

Example:
Obtain the start of frame timestamp including year, month, day, hour, minute, second, and millisecond.

```
START-TIME GET_TSTAMP_MILLI
```

NOTE
The @ (fetch) operation is not performed. Seven values are left on the stack as described in the Programmer's Reference Manual.

END–TIME (-- address)
Contains the 48 bit end of frame timestamp for data. Use with the GET_TSTAMP_MILLI or GET_TSTAMP_MICRO commands. See the START–TIME example.

BLOCK–COUNT (-- address)
Contains the block sequence number for live data. Every received frame is assigned a unique sequence number. Each side, DTE or DCE, maintains a separate set of sequence numbers. BLOCK-COUNT contains 0 and is incremented by one each time a new block is received.

REC–LENGTH (-- address)
Contains the length of the received frame. This does not include the FCS (frame check sequence) bytes.

REC-POINTER (-- address)

Contains the pointer to the frame address field (first byte) in the received frame. Since this variable contains the address of the first byte, a double fetch operation is necessary to obtain frame contents.

Example:
Obtain the second byte of the received frame (the control field).

```
REC-POINTER @  1+  C@
```

NOTE
The @ command gets the address of the first byte in the received frame. This first value is then incremented by one and one byte is fetched from the resulting address.

LEAD-NUMBER (-- address)

Contains the received lead identifier used in the test manager.

TIMER-NUMBER (-- address)

Contains the number of the expired timer. Valid values are 1 through 128.

STATUS_ERR? (-- flag)

Returns true if an error is detected in the currently processed frame. Use the following commands to detect a particular error.

Command	Error Type
OVERRUN_ERR?	Receiver overrun
CRC_ERR?	CRC error
ABORT_ERR?	Abort Error
LONG_FRM_ERR?	Frame is longer than 4096 bytes
SHORT_FRM_ERR?	Frame is shorter than 4 bytes (including 2 CRC bytes)
CTRL_BYTE_ERR?	Invalid control byte
ADDR_BYTE_ERR?	Invalid address byte
LENGTH_ERR?	Invalid frame length

Table 4-1 Error Detection

Layer 2

The layer 2 decode operation saves information concerning a frame's address, control byte and with I frames, sets up the pointer and length for the layer 3 decode.

FRAME-ADDR (-- address)

Contains the frame address field (first byte) of the received frame. Valid values are 1 or 3 for SLP, and 7 or 15 for MLP decoding.

M-CONTROL (-- address)
Contains the received frame control field. In frame modulo 8, the control field is the second byte of the received frame. In frame modulo 128, the control field is the second byte for unnumbered frames, second and third bytes for I frames and supervisory frames. See Table A-3 for possible values.

NOTE
For modulo 128, M-CONTROL contains only the second byte of I frames and supervisory frames.

FRAME-MODULO (-- address)
Contains 0 (default) for frame modulo 8, and 1 for frame modulo 128. FRAME-MODULO is set to 0 when a SABM is received, and 1 when a SABME is received (only if automatic modulo detection is selected).

M-PF (-- address)
Contains the poll/final bit for the received frame (0 or 1). In frame modulo 8, the poll/final bit is bit 5 of the control field. In frame modulo 128, the poll/final bit is bit 5 of the control field for unnumbered frames, and bit 9 of the control field in information and supervisory frames.

M-NR (-- address)
Contains the N(R) (receive sequence count) value of the received frame. Valid values are 0 through 7 for frame modulo 8, and 0 through 127 for frame modulo 128. In frame modulo 8, the N(R) values are bits 6, 7, and 8 of the control field for information and supervisory frames. In frame modulo 128, the N(R) values are bits 10 to 16 of the control field for information and supervisory frames.

M-NS (-- address)
Contains the N(S) (send sequence count) value of the received frame. Valid values are 0 through 7 for frame modulo 8, and 0 through 127 for frame modulo 128. In frame modulo 8, the N(S) values are bits 2 to 4 of the control field for information frames. In frame modulo 128, the N(S) values are bits 2 to 8 of the control field for information frames.

PKT-LENGTH (-- address)
Contains the length of the received packet. This length does not include the layer 2 bytes or the FCS bytes. If the received frame is not an I frame, the length is 0.

PKT-POINTER (-- address)
Contains the pointer to the first byte in the received packet. Since PKT-POINTER contains the address of the first byte, a double fetch operation is necessary to obtain packet values.

Example:
Obtain the first byte of the received packet.

```
PKT-POINTER  @  C@
```

NOTE
The @ command obtains the address of the packet and C@ fetches the first byte of the packet header.

Layer 3

The layer 3 decode operation saves information concerning a packet's logical group and channel number, Q and D bits, packet identifier, and fields pertinent to specific packets. Additionally, the pointer to the data field and the length of the data field is determined for data packets.

M–GFI (-- address)
Contains the GFI (general format identifier) value of the received packet. Valid values are 1 for packet modulo 8, and 2 for packet modulo 128. These are bits 5 through 8 of the first packet octet.

M–Q (-- address)
Contains the Q (qualifier) bit of a received data packet (0 or 1). This is bit 8 of the first data packet octet.

M–D (-- address)
Contains the D (delivery confirmation) bit of a received data packet, call request/incoming call, or call accepted/confirmed packet (0 or 1). This is bit 7 of the first packet octet.

M–LCG (-- address)
Contains the logical group number of the received packet. Valid values are 0 through 15. These are bits 1 through 4 of the first packet octet.

M–LCB (-- address)
Contains the logical channel number of the received packet. Valid values are 0 through 255. This is the second packet octet.

M–LCN (-- address)
Contains the combined logical group identifier and logical channel number of the received packet. Valid values are 0 through 4095. These are bits 1 through 4 of the first packet octet and all bits of the second packet octet.

M–REC–PKT–ID (-- address)
Contains the packet type identifier. This is the third packet octet. See Table A–6 for possible values.

M–RCAUSE (-- address)
Contains the cause byte. Valid values are 0 through 255. This is octet 4 of the following packets:

- Clear request/indication
- Reset request/indication
- Restart request/indication

M–RDIAG (-- address)
Contains the diagnostic byte of the received packet. Valid values are 0 through 255. This is octet 5 of the following packets:

- Clear request/indication
- Reset request/indication
- Restart request/indication

M-MORE (-- address)
Contains the more bit of a received data packet (0 or 1). Bit 5 of the third octet of data packet for modulo 8, and bit 1 of the fourth octet of data packet for modulo 128.

M-PS (-- address)
Contains the P(S) (send sequence) count of the received packet. Valid values are 0 through 7 for packet modulo 8, and 0 through 127 for packet modulo 128. The P(S) counts are bits 2 to 4 of the third octet of data packet for modulo 8, and bits 2 to 8 of the third octet of data packet for modulo 128.

M-PR (-- address)
Contains the P(R) (received sequence) count of the received packet. Valid values are 0 through 7 for packet modulo 8, and 0 through 127 for packet modulo 128. The P(R) counts are bits 6 to 8 of the third octet of data, RR, RNR, and REJ packets for modulo 8, and bits 2 to 8 of the fourth octet of data, RR, RNR, and REJ packets for modulo 128.

DATA-POINTER (-- address)
Contains the pointer to the first character of the data field in any layer 4 or user data present in a data packet. See PKT-POINTER for access examples.

DATA-LENGTH (-- address)
Contains the length of the data field in a received data packet.

M-RCALLED (-- address)
Contains a 16 byte string identifying the called address field of the received packet. See M-RCALLING for an example; string contents and handling are similar.

M-RCALLING (-- address)
Contains a 16 byte string identifying the calling address field of the received packet.

Example:
Check whether the CALLING number of the last received call request packet matches a predefined number: 43042001. The ?MATCH command is used to determine if the calling address in a received call request packet matches the defined address.

```
TCLR

#IFNOTDEF MATCH-CALL
      0  VARIABLE MATCH-CALL  12  ALLOT
#ENDIF                  ( MATCH-CALL will contain the desired calling address )

0 STATE_INIT[
      " 43042001"
      COUNT                 ( Obtain # of digits in calling address )
      15 MIN                ( Set maximum to 15 digits )
      DUP  MATCH-CALL  C!   ( Write # of digits to first byte of matching string )
      MATCH-CALL 1  +  SWAP CMOVE     ( Write calling address in matching string )
  ]STATE_INIT
```

```
0 STATE{
     R*CALLREQ 1 ?RX
     ACTION{
         M-RCALLING        ( Get this calling address )
         COUNT             ( Get # of digits in this calling address )
         MATCH-CALL        ( Get desired matching address )
         ?MATCH            ( Compare addresses )
         IF
             BEEP          ( Notify user )
             T." Address Match has occurred."  TCR
         ENDIF
     }ACTION
 }STATE
```

M-RFAC (-- address)

Contains a 128 byte string identifying the facility field of the last received call request/incoming call, call accept/connect, clear request/indication, or clear confirmation packet.

The first byte of this string contains the length of the facility field. Valid values are 0 through 63 for 1980, and 0 through 109 for 1984.

Example:
The tester receives a call request packet with a facility field that contains eight characters, i.e. the hex characters 02AA420808430303.

Obtain the length of the facility field (8 in this case).
`M-RFAC C@`

Obtain the first octet of the first facility (hex 02 in this case). This indicates:
- the first facility is a Class A facility, i.e. it is followed by a single octet parameter field; and
- the first facility is throughput class negotiation.

`M-RFAC 1 + C@`

Obtain the throughput class octet (hex AA in this case). This indicates that the throughput class for the called DTE and calling DTE is 9600 bits.
`M-RFAC 2 + C@`

Obtain the first octet of the second facility (hex 42 in this case). This indicates:
- the second facility is a Class B facility, i.e. it is followed by a 2 octet parameter field; and
- the second facility is packet size.

`M-RFAC 3 + C@`

Obtain the packet size for the called DTE (hex 08 in this case). This indicates a packet size of 256.
`M-RFAC 4 + C@`

Obtain the packet size for the calling DTE (hex 08 in this case). This indicates a packet size of 256.
`M-RFAC 5 + C@`

Obtain the first octet of the third facility (hex 43 in this case). This indicates:

- the third facility is a Class B facility, i.e. it is followed by a 2 octet parameter field; and
- the third facility is window size.

```
M-RFAC 6 + C@
```

Obtain the window size of the called DTE (hex 03 in this case). This indicates a window size of 3.

```
M-RFAC 7 + C@
```

Obtain the window size of the calling DTE (hex 03 in this case). This indicates a window size of 3.

```
M-RFAC 8 + C@
```

NOTE
Refer to CCITT X.25 (1980) Recommendation, Section 7, 'Procedure and formats for optional user facilities' for further information. Refer to CCITT X.25 (1984) Recommendation, Section 7, 'Formats for facility fields and registration fields' for further information on decoding.

M-RCUD (-- address)
Contains a 256 byte string identifying the call user data of the received packet. Valid values are 0 through 128 if fast select facility is selected, and 0 through 16 if not selected.

Example:
The tester receives a call request packet with a call user field that contains eleven characters, i.e. the hex characters C0000000003010025800064. M-RCUD can be used to obtain the following information.

Obtain the length of the call user data field (11 in this case).

```
M-RCUD C@
```

If the call user data field is present, its use and format are determined by bits 7 and 8 of the first octet.

```
M-RCUD 1 + C@
```

NOTE
Refer to CCITT 1984 Recommendation X.244 for further information on call user data.

5
CAPTURE RAM

This section describes the data flow diagram for capture to RAM and lists the commands available for test scripts. Data stored in either capture RAM or disk can be played back as described in Section 3.2. Data stored in capture RAM can be transferred to disk.

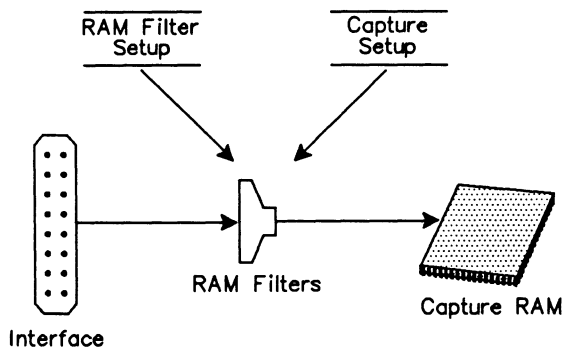


Figure 5–1 X.25 Data Flow Diagram – Capture to RAM

5.1 Capturing to RAM

CAPT_ON (--)
Saves live data in capture RAM (default).

Capture topic
Capture to RAM function key (highlighted)

CAPT_OFF (--)
Live data is not saved in capture RAM.

Capture topic
Capture to RAM function key (not highlighted)

CAPT_WRAP (--)
Initializes capture RAM so that new data overwrites (default) old data after the capture buffer is full (endless loop recording).

Capture topic
Recording Menu
→ *When Buffer Full*
WRAP function key

CAPT_FULL (--)
Initializes capture RAM so that capturing stops when the buffer is full.

Capture topic
Recording Menu
→ *When Buffer Full*
STOP function key

WARNING
CAPT_FULL and CAPT_WRAP erase all data in capture RAM.

CLEAR_CAPT (--)
Erases all data currently in capture RAM.

Capture topic
Clear function key

5.2 Transferring from RAM

Data can be transferred from capture RAM to disk, and printed as it is played back. To transfer data to disk, a data recording must be opened using the RECORD and CTOD_ON commands prior to using TRANSFER. To transfer data from capture RAM to the printer, the PRINT_ON command must first be issued. The data being transferred is displayed on the screen.

TRANSFER (--)
Transfers data from the selected data source.

Capture topic
Save RAM to Disk function key (highlighted)

QUIT_TRA (--)
Abruptly terminates the transfer of data from capture RAM to disk.

Capture topic
Save RAM to Disk function key (not highlighted)

TRA_ALL (--)
Transfers the entire contents of capture RAM (default) when the TRANSFER command is used.

Capture topic
Save RAM to Disk function key
All function key

TRA_START (--)
Selects the starting block for transfer and is used with TRA_END when a partial transfer is desired. Use the cursor keys to locate the desired starting block prior to calling TRA_START. TRA_START selects the last scrolled block as the initial starting block for transfer.

Capture topic
Save RAM to Disk function key
Set Start function key

TRA_END (--)
Selects the final block for transfer and is used with TRA_START when a partial transfer is desired. Use the cursor keys to locate the desired final block prior to calling TRA_END. TRA_END selects the last scrolled block as the final starting block for transfer.

Capture topic
Save RAM to Disk function key
Set End function key

SEE_TRA (--)
Displays the timestamps for the initial and final blocks selected for transfer in the Command and Test Script Windows.

Example:
Open a data file with the filename 'DATA1' and transfer all data from capture RAM to disk. After the transfer is complete, turn off data recording.

```
FROM_CAPT                             ( Designate Capture RAM as data source )
HALT                                  ( Enter playback mode )
" DATA1" =TITLE                       ( Assign filename DATA1 )
RECORD                                ( Open data recording )
CTOD_ON                               ( Enable Capture Transfer to disk )
TRA_ALL                               ( Transfer all data )
TRANSFER                              ( Transfer data from Capture to disk )
DISK_OFF                              ( Turn off data recording )
```

To Disk

CTOD_ON (--)
Enables transfer of data from capture RAM to disk when data source is playback RAM and a data recording file is open.

CTOD_OFF (--)
Disables transfer of data from capture RAM to disk (default) when data source is playback RAM.

To Printer

PRINT_ON (--)
Prints data lines as displayed during playback from either capture RAM or disk. No printout is made when the source is live data. The printer must be configured from the Printer Port Setup Menu under the **Setup** topic on the Home processor.

Print topic
Print On function key

PRINT_OFF (--)
Data is not printed during playback (default).

Print topic
Print Off function key

Example:
Transfer all data from capture RAM to the printer.

```
FROM_CAPT                    ( Designate Capture RAM as data source )
HALT                         ( Enter playback mode )
PRINT_ON                     ( Enable printing )
TRA_ALL                      ( Transfer all )
TRANSFER                     ( Transfer data to printer )
```

6
DISK RECORDING

Live data from the interface can be recorded to either a floppy or hard disk. Data stored in either capture RAM or disk can be played back as described in Section 3.2. Data stored in capture RAM can be transferred to disk as described in Section 5.2.

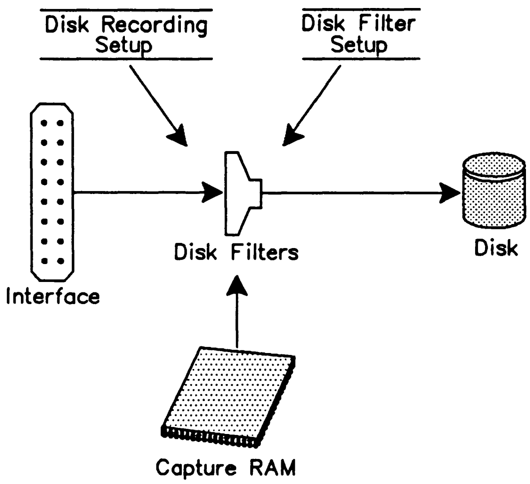


Figure 6–1 X.25 Data Flow Diagram – Recording to Disk

DISK_WRAP (--)
Selects disk recording overwrite (default).

Capture topic
Recording Menu
→ *When File Full*
WRAP function key

DISK_FULL (--)
Turns off disk recording overwrite. Recording continues until the data recording file is full.

Capture topic
Recording Menu
→ *When File Full*
STOP function key

WARNING
DISK_WRAP and DISK_FULL must be called prior to opening a recording with the RECORD command. If called while recording is in process, the status of the disk recording overwrite for this recording session will not change.

RECORD (--)

Opens a data recording file. When used in the Command Window, the filename can be specified as part of the command.

Example:
```
RECORD DATA1
```

Capture topic
Record to Disk function key (highlighted)

NOTE
When RECORD is used in a test script, the filename must be specified with =TITLE. Because of the relatively long time required to open a disk file (especially on a floppy drive), RECORD should not be used within time critical portions of a test script.

Trace report lines are included in the data file when an application requests start and end recording. The information in these traces identifies the traffic type and application program used while the data was being recorded.

Example:
```
Recording Start : X.25 Monitor     WAN Port 1 RS232-C
V1.3-1.3                           PT500 - 19     SN# 01-261

Recording End   : X.25 Monitor     WAN Port 1 RS232-C
V1.3-1.3                           PT500 - 19     SN# 01-261
```

DISK_OFF (--)

Live data is not recorded to disk. The current disk recording is closed.

Capture topic
Record to Disk function key (not highlighted)

NOTE
Refer to the Programmer's Reference Manual for multi-processor disk recording.

DIS_REC (--)

Momentarily suspends data recording. The data recording file remains open but no data is saved to disk.

Capture topic
Record to Disk function key (highlighted)
Suspend Recording function key (highlighted)

ENB_REC (--)

Enables data recording. The data recording file remains open and live data is recorded to disk.

Capture topic
Record to Disk function key (highlighted)
Suspend Recording function key (not highlighted)

7
DISPLAY FORMAT

The X.25 Monitor and Emulation applications can display data from the line (live data), from capture RAM, or from a disk recording in the following display formats:

- Hexadecimal
- Character
- Short
- Complete
- Split
- Trace Statements

The data flow diagram for displaying and printing data, as well as commands available for test scripts, are described in this section.

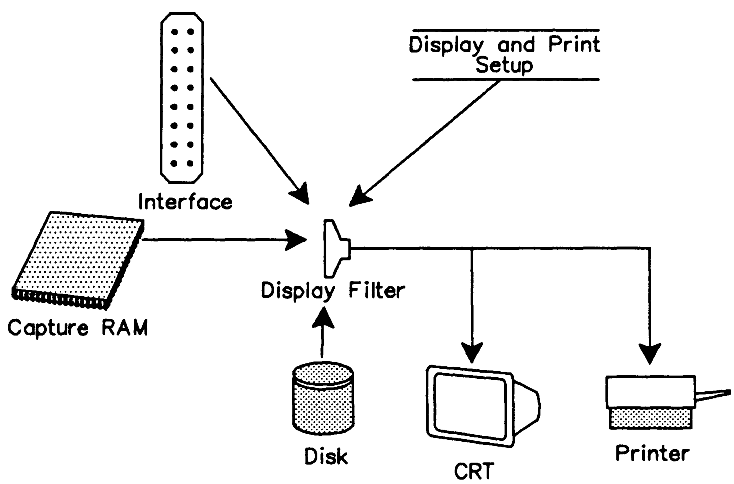


Figure 7–1 X.25 Data Flow Diagram – Display and Print

NOTE
Data can only be printed in playback mode.

```
                         Display Format Menu

 -> Display Format COMPLETE      Dual Window             OFF

    Timestamp      OFF           Trace Display Format    SHORT
    Character Set  ASCII
    Frame Layer    TEXT          Throughput Graph        OFF
    Packet Layer   TEXT            Short Interval (sec)  10
    Data Field     CHARACTER       Long  Interval (sec)  600
```

Figure 7–2 Display Format Menu

→ Display Format
The default display is short format. *Frame Layer, Packet Layer,* and *Data Field* can only be modified when *Display Format* is set to *COMPLETE.*

REP_ON (--)
Turns on data display (default).

OFF function key (not highlighted)

REP_OFF (--)
Turns off data display.

OFF function key (highlighted)

REP_COMP (--)
Displays data in a comprehensive report. Each protocol layer has its own display generator and thus can be turned on, off, or selected as text, hex, or character display.

COMPLETE function key

REP_SHORT (--)
Displays data in a condensed report (default). This includes the source identifier, frame address, frame type, LCN value, packet type, and for data packets, the length and the first 50 characters of the data field. This format is useful for higher speed monitoring as more frames per screen are displayed and processing is kept to a minimum.

SHORT function key

NOTE
The length of the data packet reported can be changed using the SET_SREP_LEN command.

REP_HEX (--)
Displays timestamps or block sequence numbers and the port identifier in text. Frame contents are displayed in hex.

HEX function key

REP_CHAR (--)
Displays timestamps or block sequence numbers and the port identifier in text. Frame contents are displayed in the currently selected character set.

CHARACTER function key

SPLIT_ON (--)
Displays data in short format with a split screen display. The screen is divided in half with frames sent from the DCE displayed on the left and frames sent from the DTE on the right.

SPLIT function key

NOTE
Only the first 38 characters of a trace statement are displayed when split display format is selected.

NOTE
The length of the data packet reported can be changed using the SET_SREP_LEN command.

SPLIT_OFF (--)
Sets the data display to the full screen short format display (default).

SHORT function key

SET_SREP_LEN (length --)
Specifies the length of the data packet reported. *Valid values are 0 through 50 (default) for REP_SHORT, and 0 through 10 (default) for REP_SPLIT.*

REP_NONE (--)
Displays only trace statements.

TRACE function key

→ *Timestamp*
Timestamp reporting is available when the display format is not in short or split mode.

TIME_OFF (--)
Timestamps are not displayed (default). Block sequence numbers are displayed for each received frame.

OFF function key

TIME_ON (--)
Displays the start and end of frame timestamps as minutes, seconds, and tenths of milliseconds. Block sequence numbers for received frames are not displayed.

MM:SS.ssss function key

TIME_DAY (--)
Displays the start and end of frame timestamps as days, hours, minutes, and seconds. Block sequence numbers for received frames are not displayed.

DD HH:MM:SS function key

→ *Character Set*
Selects the character set for data display.

R=ASCII (--)
Sets the character set for data display to ASCII (default).

ASCII function key

R=EBCDIC (--)
Sets the character set for data display to EBCDIC.

EBCDIC function key

R=HEX (--)
Sets the character set for data display to hex.

HEX function key

R=JIS8 (--)
Sets the character set for data display to JIS8.

JIS8 function key

→ *Frame Layer*

FRM_ON (--)

Displays layer 2 data in a detailed report (default).

TEXT function key

FRM_OFF (--)

Layer 2 data is not displayed.

OFF function key

FRM_HEX (--)

Displays layer 2 data in hex.

HEX function key

FRM_CHAR (--)

Displays layer 2 data in the currently selected character set.

CHARACTER function key

→ *Packet Layer*

PKT_ON (--)

Displays layer 3 data in a detailed report (default).

TEXT function key

PKT_OFF (--)

Layer 3 data is not displayed.

OFF function key

PKT_HEX (--)

Displays layer 3 data in hex.

HEX function key

PKT_CHAR (--)

Displays layer 3 data in the currently selected character set.

CHARACTER function key

→ *Data Field*

DATA_ON or **DATA_CHAR** (--)
Displays the data field of data packets in the currently selected character set (default).

CHARACTER function key

DATA_OFF (--)
The data field of data packets is not displayed.

OFF function key

DATA_HEX (--)
Displays the data field of data packets in hex.

HEX function key

FAC_IN_COMP (--)
Displays the facility field in a detailed X.25 protocol report (default).

NOTE
Display Format must be set to COMPLETE.

FAC_IN_HEX (--)
Displays the entire facility field in hex.

NOTE
Display Format must be set to COMPLETE.

CLEAR_CRT (--)
Clears the display in the Data Window.

Display topic
Clear function key

→ *Dual Window*
If two applications have been loaded, the screen can be divided horizontally to display data from both applications. The current application is always displayed in the top window.

FULL (--)
Uses the entire Data Display Window for the current application.

Dual window commands vary depending on the machine configuration. Table 7-1 shows the relationship between machine configuration, application processors, and dual window commands.

Machine Type	Command	Dual Window AP #	
WAN/WAN	DUAL_1+2	AP #1	AP #2
BRA/WAN	DUAL_1+2	AP #1	AP #2
	DUAL_1+7	AP #1	AP #3
	DUAL_2+7	AP #2	AP #3
PRA	DUAL_3+4	AP #1	AP #2
PRA/BRA/WAN	DUAL_1+2	AP #1	AP #2
	DUAL_1+3	AP #1	AP #4
	DUAL_1+4	AP #1	AP #5
	DUAL_1+7	AP #1	AP #3
	DUAL_2+3	AP #2	AP #4
	DUAL_2+4	AP #2	AP #5
	DUAL_2+7	AP #2	AP #3
	DUAL_3+4	AP #4	AP #5
	DUAL_3+7	AP #4	AP #3
	DUAL_4+7	AP #5	AP #3
BRA/BRA	DUAL_1+2	AP #1	AP #2
	DUAL_1+3	AP #1	AP #4
	DUAL_1+4	AP #1	AP #5
	DUAL_1+5	AP #1	AP #6
	DUAL_1+7	AP #1	AP #3
	DUAL_2+3	AP #2	AP #4
	DUAL_2+4	AP #2	AP #5
	DUAL_2+5	AP #2	AP #6
	DUAL_2+7	AP #2	AP #3
	DUAL_3+4	AP #4	AP #5
	DUAL_3+5	AP #4	AP #6
	DUAL_3+7	AP #4	AP #3
	DUAL_4+5	AP #5	AP #6
	DUAL_4+7	AP #5	AP #3
	DUAL_5+7	AP #6	AP #3
PRA/WAN	DUAL_1+3	AP #1	AP #2
	DUAL_1+4	AP #1	AP #3
	DUAL_3+4	AP #2	AP #3

Table 7-1 Dual Window Commands

→ *Trace Display Format*
Selects the display format for trace statements.

TRACE_SHORT (--)
Displays the trace statement on one line (short format) containing only user-defined text.

SHORT function key

TRACE_COMP (--)
Displays the trace statement on two lines (complete format). Block sequence numbers or timestamps are displayed on the first line, and user-defined text on the second line.

COMPLETE function key

→ *Throughput Graph*
The throughput rate can be calculated, displayed as a bar graph, and printed out. The X.25 Monitor calculates throughput by counting the number of bytes on each side of the line during two intervals - one short, one long. This figure is divided by the time interval to arrive at a bits per second figure for each time interval (for both DTE and DCE data).

NOTE
For accurate throughput measurement, the bit rate (line speed) must be set on the Monitor/Emulation Configuration Menu or in the INTERFACE-SPEED variable to match the actual line speed.

The baud rate, as stored in the INTERFACE-SPEED variable, is used to calculate a percentage throughput based on theoretical limits.

INTERFACE-SPEED (-- address)
Contains the current bit rate (default value is 64000).

Example:
Set the throughput measurement speed to 2400.

```
2400 INTERFACE-SPEED !
TPR_ON
```

TPR_ON (--)
Calculates and displays the throughput rate as a bar graph.

DISPLAY function key

WARNING
If the short interval, long interval, or speed is changed, TPR_ON must be called after the changes are made.

TPR_OFF (--)
The throughput rate is not calculated or displayed.

OFF function key

PRINT_TPR (--)
Calculates and displays the throughput rate as a bar graph, and prints the long term interval measurements.

DISPLAY AND PRINT function key

→ *Short Interval*
Sets the short time interval, in seconds, for measuring, displaying, and printing the throughput results.

SHORT-INTERVAL (-- address)
Contains the current duration of the short interval (default value is 10 seconds).

Example:
Set the short interval to 20 seconds.

```
20 SHORT-INTERVAL !
TPR_ON
```

Modify Short Interval function key

→ *Long Interval*
Sets the long time interval, in seconds, for measuring, displaying, and printing the throughput results.

LONG-INTERVAL (-- address)
Contains the current duration of the long interval (default value is 600 seconds).

Example:
Set the long interval to 300 seconds.

```
300 LONG-INTERVAL !
TPR_ON
```

Modify Long Interval function key

8
FILTERS

Filters provide the capability of passing or blocking specific events from the display, capture RAM, or disk recording. These three filters act independently. This section describes the commands used to pass or block frames/packets, and activate or deactivate each of the three filters.

```
                              Filter Setup Menu

    Filter Type      DISPLAY          Trace Statements  ON
    Filter Status    DEACTIVATED   -> Selective Address 4034624545
    Lead Changes     BLOCK            Selective LCN #1  ---        LCN #2 ---
                                      Selective LCN #3  ---        LCN #4 ---

  Frame Layer:
    SABM    PASS     I       PASS     UA          PASS   DM            PASS
    SABME   PASS     RR      PASS     DISC        PASS   FRMR/CMDR     PASS
    SARM    PASS     RNR     PASS     REJ         PASS   Invalid       PASS

  Packet Layer:
    Call    PASS     RR      PASS     Restart     PASS   Registration  PASS
    Clear   PASS     RNR     PASS     Reset       PASS   Diagnostic    PASS
    Data    PASS     REJ     PASS     Interrupt   PASS   Invalid       PASS
```

Figure 8–1 Filter Setup Menu

→ *Filter Type*
There are three separate filter processes which act independently of each other: *DISPLAY, RAM,* and *DISK*.

→ *Filter Status*
Filters can be deactivated (default) or activated at any time. When the filter status is changed, the connection diagram changes to reflect this. Figure 8–2 shows live data as the data source with display filters activated. If deactivated, all lead changes, trace statements, frames, and packets go to display.

NOTE
Lead changes are detected only if leads are enabled.

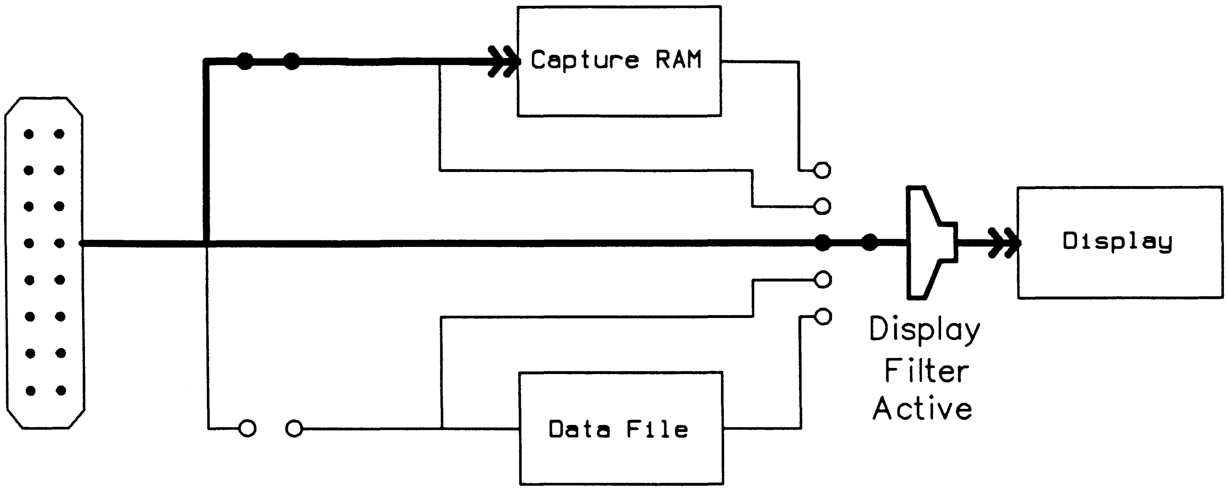


Figure 8–2 Connection Diagram – Display Filters Activated

REP_FILTER_ON (--)
Activates the display filter.

→ *Filter Type*
DISPLAY function key
→ *Filter Status*
ACTIVATED function key

REP_FILTER_OFF (--)
Deactivates the display filter.

→ *Filter Type*
DISPLAY function key
→ *Filter Status*
DEACTIVATED function key

RAM_FILTER_ON (--)
Activates the capture RAM filter.

→ *Filter Type*
RAM function key
→ *Filter Status*
ACTIVATED function key

RAM_FILTER_OFF (--)
Deactivates the capture RAM filter.

→ *Filter Type*
RAM function key
→ *Filter Status*
DEACTIVATED function key

DISK_FILTER_ON (--)
Activates the disk filter.

→ *Filter Type*
DISK function key
→ *Filter Status*
ACTIVATED function key

DISK_FILTER_OFF (--)
Deactivates the disk filter.

→ *Filter Type*
DISK function key
→ *Filter Status*
DEACTIVATED function key

→ *Lead Changes*
Lead changes can be blocked (default) or passed.

R1=ALL (--)
Passes lead changes to the display.

→ *Filter Type*
DISPLAY function key
→ *Lead Changes*
PASS function key

R1=NONE (--)
Blocks lead changes from the display.

→ *Filter Type*
DISPLAY function key
→ *Lead Changes*
BLOCK function key

C1=ALL (--)
Passes lead changes to capture RAM.

→ *Filter Type*
RAM function key
→ *Lead Changes*
PASS function key

C1=NONE (--)
Blocks lead changes from capture RAM.

→ *Filter Type*
RAM function key
→ *Lead Changes*
BLOCK function key

D1=ALL (--)
Passes lead changes to disk.

→ *Filter Type*
DISK function key
→ *Lead Changes*
PASS function key

D1=NONE (--)
Blocks lead changes from disk.

→ *Filter Type*
DISK function key
→ *Lead Changes*
BLOCK function key

→ *Trace Statements*
Trace statements can be blocked or passed (default).

YES RTRACE (--)
Passes trace statements to the display.

→ *Filter Type*
DISPLAY function key
→ *Trace Statements*
ON function key

NO RTRACE (--)
Blocks trace statements from the display.

→ *Filter Type*
DISPLAY function key
→ *Trace Statements*
OFF function key

YES CTRACE (--)
Passes trace statements to capture RAM.

→ *Filter Type*
RAM function key
→ *Trace Statements*
ON function key

NO CTRACE (--)
Blocks trace statements from capture RAM.

→ *Filter Type*
RAM function key
→ *Trace Statements*
OFF function key

YES DTRACE (--)
Passes trace statements to disk.

→ *Filter Type*
DISK function key
→ *Trace Statements*
ON function key

NO DTRACE (--)
Blocks trace statements from disk.

→ *Filter Type*
DISK function key
→ *Trace Statements*
OFF function key

→ *Selective Address*
If filters are activated and a specific address has been entered, no frames are displayed until a call request/incoming call packet with that called or calling address is received. The call request/incoming call is shown and all subsequent frames and packets with a PASS status, received on that LCN, are displayed until a clear request/indication packet on that LCN is received. After the clear request/indication packet, no further frames or packets are displayed until another call request/incoming call containing the specified address is received.

NOTE
If a second call request/incoming call, containing the specified called/calling address, is received on a different LCN, only traffic on the new LCN is displayed.

NOTE
If a selective address is specified, the selective LCN's cannot be used.

RTRIG_ON (--)
Turns off the display. If display filters are activated, the display remains off until a call request/incoming call packet is received containing the address specified. All packets on the same LCN are then displayed until a clear request/indication packet is received on the same LCN. After this packet is shown, the display is turned off again. If filters are not activated with the REP_FILTER_ON command, no frames or packets are displayed.

→ *Filter Type*
DISPLAY function key
→ *Selective Address*
ONE function key
→ *Filter Status*
ACTIVATED function key

CTRIG_ON (--)
Activates call address filtering for RAM filters. See description under RTRIG_ON.

→ *Filter Type*
RAM function key
→ *Selective Address*
ONE function key
→ *Filter Status*
ACTIVATED function key

DTRIG_ON (--)
Activates call address filtering for disk filters. See description under RTRIG_ON.

→ *Filter Type*
DISK function key
→ *Selective Address*
ONE function key
→ *Filter Status*
ACTIVATED function key

RTRIG_OFF (--)
Turns on the display and passes all addresses.

→ *Filter Type*
DISPLAY function key
→ *Selective Address*
ALL function key

CTRIG_OFF (--)
Turns on capture RAM and captures all addresses.

→ *Filter Type*
RAM function key
→ *Selective Address*
ALL function key

DTRIG_OFF (--)
Enables the disk recording and records all addresses.

→ *Filter Type*
DISK function key
→ *Selective Address*
ALL function key

R-WCALLED (-- address)

Contains a 16 byte string identifying the display selective address. The first byte of the string contains the length of the address.

Example:
Set the display filter selective address to 43042001.

```
" 43042001"                  ( Define the selective address )
COUNT 15 MIN                 ( No more than 15 characters allowed )
DUP R-WCALLED C!             ( Store count at first byte )
R-WCALLED 1+ SWAP CMOVE      ( Move characters to R-WCALLED )
RTRIG_ON                     ( Use Display selective address filter )
REP_FILTER_ON                ( Activate Display filter )
```

→ *Filter Type*
DISPLAY function key
→ *Selective Address*
ONE function key
→ *Filter Status*
ACTIVATED function key

NOTE
Use R-WCALLED to define a display filter selective address within a test script.

C-WCALLED (-- address)

Contains a 16 byte string identifying the capture RAM filter selective address. The first byte of the string contains the length of the address. See example under R-WCALLED.

D-WCALLED (-- address)

Contains a 16 byte string identifying the disk filter selective address. The first byte of the string contains the length of the address. See example under R-WCALLED.

→ *Selective LCN*

If display filters are activated and a specific LCN is entered, only packets that have a PASS status on that LCN are displayed. Up to four LCN's can be selected.

NOTE
Commands for display filters and selective LCN #1 are described here as an example. For a complete list of commands, see Table 8-1.

RLCN=ALL (--)

Passes packets, on all LCN's, to the display.

→ *Filter Type*
DISPLAY function key
→ *Selective LCN #1*
ALL function key

=RLCN1 (LCN value --)
Specifies the logical channel for which packets are passed to the display.

Example:
Specify logical channel number 5 for Selective LCN #1.

```
5 =RLCN1
```

→ *Filter Type*
DISPLAY function key
→ *Selective LCN #1*
Modify function key

RLCN1=SEL (--)
Passes packets to the display on the specified logical channel, as defined with the =RLCN1 command (default value is 0).

Example:
Set Selective LCN #1 to pass all packets on logical channel 20.

```
20 =RLCN1       ( Define selective LCN )
RLCN1=SEL       ( Use selective LCN#1 )
```

→ *Filter Type*
DISPLAY function key
→ *Selective LCN #1*
Select function key

RLCN1=OFF (--)
Selective LCN #1 is not used for display filters.

→ *Filter Type*
DISPLAY function key
→ *Selective LCN #1*
OFF function key

Description	Display	RAM	Disk
All LCNs passed	RLCN=ALL	CLCN=ALL	DLCN=ALL
Sets Selective LCN #1	=RLCN1	=CLCN1	=DLCN1
Uses Selective LCN #1 for filters	RLCN1=SEL	CLCN1=SEL	DLCN1=SEL
Selective LCN #1 not used for filters	RLCN1=OFF	CLCN1=OFF	DLCN1=OFF
Sets Selective LCN #2	=RLCN2	=CLCN2	=DLCN2
Uses Selective LCN #2 for filters	RLCN2=SEL	CLCN2=SEL	DLCN2=SEL
Selective LCN #2 not used for filters	RLCN2=OFF	CLCN2=OFF	DLCN2=OFF
Sets Selective LCN #3	=RLCN3	=CLCN3	=DLCN3
Uses Selective LCN #3 for filters	RLCN3=SEL	CLCN3=SEL	DLCN3=SEL
Selective LCN #3 not used for filters	RLCN3=OFF	CLCN3=OFF	DLCN3=OFF
Sets Selective LCN #4	=RLCN4	=CLCN4	=DLCN4
Uses Selective LCN #4 for filters	RLCN4=SEL	CLCN4=SEL	DLCN4=SEL
Selective LCN #4 not used for filters	RLCN4=OFF	CLCN4=OFF	DLCN4=OFF

Table 8–1 Selective Filter Commands

Frame Layer:
If display filters are activated, any frame with a PASS status is displayed unless the selective address has been set to a specific address. See the description under *Selective Address*.

NOTE
Commands for display filters and SABM frames are described here as an example. For a complete list of commands, see Table 8–2.

R2+SABM (--)
Passes SABM frames to the display.

→ *Filter Type*
DISPLAY function key
→ *SABM*
PASS function key

R2–SABM (--)
Blocks SABM frames from the display.

→ *Filter Type*
DISPLAY function key
→ *SABM*
BLOCK function key

R2=ALL (--)
Passes all frames (default) to the display.

→ *Filter Type*
DISPLAY function key
→ *SABM*
ALL FRAMES function key

R2=NONE (--)
Blocks all frames and packets from the display.

→ *Filter Type*
DISPLAY function key
→ *SABM*
NO FRAMES function key

Description		Display	RAM	Disk
All	Pass (default)	R2=ALL	C2=ALL	D2=ALL
	Block	R2=NONE	C2=NONE	D2=NONE
Set Asynchronous Balanced Mode	Pass	R2+SABM	C2+SABM	D2+SABM
	Block	R2-SABM	C2-SABM	D2-SABM
Set Asynchronous Response Mode (LAP procedure)	Pass	R2+SARM	C2+SARM	D2+SARM
	Block	R2-SARM	C2-SARM	D2-SARM
Disconnect	Pass	R2+DISC	C2+DISC	D2+DISC
	Block	R2-DISC	C2-DISC	D2-DISC
Unnumbered Acknowledgement	Pass	R2+UA	C2+UA	D2+UA
	Block	R2-UA	C2-UA	D2-UA
Disconnect Mode	Pass	R2+DM	C2+DM	D2+DM
	Block	R2-DM	C2-DM	D2-DM
Frame Reject	Pass	R2+FRMR	C2+FRMR	D2+FRMR
	Block	R2-FRMR	C2-FRMR	D2-FRMR
Command Reject (LAP procedure)	Pass	R2+CMDR	C2+CMDR	D2+CMDR
	Block	R2-CMDR	C2-CMDR	D2-CMDR
Information	Pass	R2+I	C2+I	D2+I
	Block	R2-I	C2-I	D2-I
Receive Ready	Pass	R2+RR	C2+RR	D2+RR
	Block	R2-RR	C2-RR	D2-RR
Receive Not Ready	Pass	R2+RNR	C2+RNR	D2+RNR
	Block	R2-RNR	C2-RNR	D2-RNR
Reject	Pass	R2+REJ	C2+REJ	D2+REJ
	Block	R2-REJ	C2-REJ	D2-REJ
Invalid	Pass	R2+INV	C2+INV	D2+INV
	Block	R2-INV	C2-INV	D2-INV

Table 8–2 Frame Layer Filter Commands

Packet Layer:
If display filters are activated, any packet with a PASS status is displayed unless the selective address or a selective LCN has been set to a specific value. See the description under *Selective Address* and *Selective LCN.*

NOTE
If the I frame has been blocked, a filter at the packet layer is inappropriate. This is indicated by the dashes shown beside the individual packet layer items.

NOTE
Commands for display filters and call packets are described here as an example. For a complete list of commands, see Table 8–3.

R3+CALL (--)
Passes call request/incoming call and call connected/accepted packets to the display.

→ *Filter Type*
DISPLAY function key
→ *Call*
PASS function key

R3–CALL (--)
Blocks call request/incoming call and call connected/accepted packets from the display.

→ *Filter Type*
DISPLAY function key
→ *Call*
BLOCK function key

R3=ALL (--)
Passes all packets (default) to the display.

→ *Filter Type*
DISPLAY function key
→ *Call*
ALL PACKETS function key

R3=NONE (--)
Blocks all packets from the display.

→ *Filter Type*
DISPLAY function key
→ *Call*
NO PACKETS function key

Description		Display	RAM	Disk
All	Pass (default)	R3=ALL	C3=ALL	D3=ALL
	Block	R3=NONE	C3=NONE	D3=NONE
Call Request Call Connect	Pass	R3+CALL	C3+CALL	D3+CALL
	Block	R3-CALL	C3-CALL	D3-CALL
Clear Request Clear Confirmation	Pass	R3+CLEAR	C3+CLEAR	D3+CLEAR
	Block	R3-CLEAR	C3-CLEAR	D3-CLEAR
Reset Request Reset Confirmation	Pass	R3+RESET	C3+RESET	D3+RESET
	Block	R3-RESET	C3-RESET	D3-RESET
Restart Request Restart Confirmation	Pass	R3+RESTART	C3+RESTART	D3+RESTART
	Block	R3-RESTART	C3-RESTART	D3-RESTART
Interrupt Interrupt Confirmation	Pass	R3+INT	C3+INT	D3+INT
	Block	R3-INT	C3-INT	D3-INT
Data	Pass	R3+DATA	C3+DATA	D3+DATA
	Block	R3-DATA	C3-DATA	D3-DATA
Receive Ready	Pass	R3+RR	C3+RR	D3+RR
	Block	R3-RR	C3-RR	D3-RR
Receive Not Ready	Pass	R3+RNR	C3+RNR	D3+RNR
	Block	R3-RNR	C3-RNR	D3-RNR
Reject	Pass	R3+REJ	C3+REJ	D3+REJ
	Block	R3-REJ	C3-REJ	D3-REJ
Registration Request Registration Confirmation	Pass	R3+REG	C3+REG	D3+REG
	Block	R3-REG	C3-REG	D3-REG
Diagnostic	Pass	R3+DIAG	C3+DIAG	D3+DIAG
	Block	R3-DIAG	C3-DIAG	D3-DIAG
Invalid	Pass	R3+INV	C3+INV	D3+INV
	Block	R3-INV	C3-INV	D3-INV

Table 8-3 Packet Layer Filter Commands

9
EMULATION CONFIGURATION

This section displays the Emulation Configuration and LCN Setup Menus and describes commands corresponding to each item.

```
                           Setup Menu

 → Protocol Standard      X.25(1984)
   Emulation Mode         DTE            Link Procedure      SINGLE LINK

 Physical Layer:
   Emulation Interface    TO DCE         Bit Rate            UNKNOWN
   Interface Type         RS232C/V.28    Interface Leads     DISABLED
   Interframe Fill        FLAG           External Tx Clock   OFF
```

Figure 9–1 Setup Menu

The emulation is active at all times, i.e. automatic responses to protocol events are generated even during parameter setup. However, most of the commands which change the physical layer require re–initialization of the program. This is accomplished using the STARTUP command.

STARTUP (--)
Initializes the X.25 Emulation to the disconnect state in layer 2 and the idle state in layer 3, and configures the physical interface.

NOTE
Use STARTUP once after all physical layer changes are made.

9.1 Protocol Standard

→ *Protocol Standard*
Selects a protocol standard for emulation.

STD=NONE (--)
Conforms to a combination of the CCITT X.25 (1980/1984) Recommendations. The behaviour can be modified by the user.

NONE function key

STD=X25(80) (--)
Conforms to the CCITT X.25 (1980) Recommendation.

X.25(1980) function key

STD=X25(84) (--)
Conforms to the CCITT X.25 (1984) Recommendation (default).

X.25(1984) function key

The following table shows the effects of the Protocol Standard on the emulation parameters.

Parameters/Procedures	NONE	X.25(1980)	X.25(1984)
Link Procedure	Single Link or Multilink	Single Link	Single Link or Multilink
Sequence Number	Modulo 8 or 128	Modulo 8	Modulo 8 or 128
Poll Bit for SABM, SABME, and DISC	P=0 or P=1	P=0	P=0 or P=1
Unsolicited DM	Supported or Not Supported	Not Supported	Supported
Maximum User Data in DATA Packet	4096 Octets	1024 Octets	4096 Octets
Facility Field Length	109 Octets	63 Octets	109 Octets
Call User Data	16 Octets 128 Octets with fast select facility	16 Octets	16 Octets 128 Octets with fast select facility
Call Connected Format	Basic or Extended	Basic	Basic or Extended
Clear Request Format	Basic or Extended	Basic	Basic or Extended
Clear Confirmation Format	Basic or Extended	Basic	Basic or Extended
Maximum Interrupt Data	32 Octets	1 Octet	32 Octets

Table 9–1 Emulation Protocol Standard Parameters/Procedures

9.2 Emulation Mode

→ *Emulation Mode*
Selects the logical type of emulation and determines the value of the address byte for commands and responses.

DTE_END (--)
Selects a logical DTE emulation mode (default). For single link procedure, the command address equals 01 and the response address equals 03. For multilink procedure, the command address equals 07 and the response address equals 0F.

DTE function key

DCE_END (--)
Selects a logical DCE emulation mode. For single link procedure, the command address equals 03 and the response address equals 01. For multilink procedure, the command address equals 0F and the response address equals 07.

DCE function key

9.3 Link Procedure

→ *Link Procedure*
Selects single link or multilink emulation and sets the value of the address bytes for commands and responses.

SLP (--)
Selects single link procedure.

Example:
```
SLP SET_FADR
```

SINGLE LINK function key

MLP (--)
Selects multilink procedure.

Example:
```
MLP SET_FADR
```

MULTILINK function key

SET_FADR (--)
Sets command and response addresses for single/multilink and DTE/DCE logical emulation mode.

	Single Link Addresses		Multilink Addresses	
	Command	Response	Command	Response
DTE Emulation	01	03	07	0F
DCE Emulation	03	01	0F	07

Table 9–2 Command Response Addresses

9.4 Physical Layer

→ *Emulation Interface*
Selects the physical type of emulation.

NOTE
Refer to Table 9–3 for clocking selections depending on the emulation interface.

TO_DCE_IF (--)
Selects the 'to DCE' interface (default).

TO DCE function key

TO_DTE_IF (--)
Selects the 'to DTE' interface.

TO DTE function key

→ *Interface Type*
IF=V28 (--)
Selects the V.28/RS–232C connector (default) and electrically isolates the other connectors on the port.

RS232C/V.28 function key

IF=V11 (--)
Selects the V.11/X.21 connector and electrically isolates the other connectors on the port.

RS422/V.11 function key

IF=V35 (--)
Selects the V.35 connector and electrically isolates the other connectors on the port.

V.35 function key

IF=V36 (--)
Selects the V.36 (RS-449) connector and electrically isolates the other connectors on the port.

RS449/V.36 function key

NOTE
A WAN tester has a V.28, V.11, and either a V.35 or V.36 connector. These commands are only applicable when the program is running on a WAN interface.

→ *Interframe Fill*
Selects the bit pattern transmitted between blocks of data.

INTERFRAME-FILL (-- address)
Contains the current interframe fill character.

Examples:
Set interframe fill to MARK.
```
MRK  INTERFRAME-FILL  !
```

MARK function key

Set interframe fill to FLAG (default).
```
SYNC  INTERFRAME-FILL !
```

FLAG function key

→ *Bit Rate*
The interface speed can be selected from preset values on the Interface Port Speed Menu, set to a user-defined speed, or measured depending on the emulation interface and clocking selections.

External Tx Clock	TO DTE	TO DCE
OFF	Select	Measure
ON	Measure	Select

Table 9-3 Effect of Clocking and Emulation Interface Selections on Bit Rate

NOTE
Clocking is provided by the attached equipment when the bit rate can be selected.

INTERFACE-SPEED (-- address)
Contains the current bit rate (default is 64000).

Example:
Set the interface speed to 9600.
```
9600  INTERFACE-SPEED !
```

NOTE
Integer values must be written to INTERFACE-SPEED. Thus, to obtain a bit rate of 134.5, either 134 or 135 can be written to INTERFACE-SPEED.

→ *Interface Leads*
Individual or all interface leads can be enabled or disabled (default). Leads must be enabled for test manager detection.

ENABLE_LEAD (lead identifier --)
Enables the specified lead. Refer to the Programmer's Reference Manual for a list of supported leads for each interface type.

Example:
Enable the request to send lead.

```
IRS ENABLE_LEAD
```

DISABLE_LEAD (lead identifier --)
Disables (default) the specified lead. Refer to the Programmer's Reference Manual for a list of supported leads for each interface type.

Example:
Disable the clear to send lead.

```
ICS DISABLE_LEAD
```

ALL_LEADS (-- lead identifier)
Enables/disables all leads supported on the currently selected WAN interface. ALL_LEADS must be used with ENABLE_LEAD or DISABLE_LEAD.

Example 1:
Enable all leads for the current interface.

```
ALL_LEADS ENABLE_LEAD
```

ENABLED function key

Example 2:
Disable all leads for the current interface.

```
ALL_LEADS DISABLE_LEAD
```

DISABLED function key

→ *External Tx Clock*
EXT_CLOCK (--)
Clocking is provided by the DTE.

ON function key

STD_CLOCK (--)
Clocking is provided by the DCE (default).

OFF function key

The following sequence illustrates the use of the configuration commands. STARTUP is only called at the end of the configuration sequence.

```
IF=V35                          ( Use V.35 test connector )
DTE_END
TO_DCE_IF                       ( Set for full DTE simulation )
SYNC  INTERFRAME-FILL !         ( Use sync - 7E - between frames )
56000 INTERFACE-SPEED !         ( Set baud rate )
STD_CLOCK                       ( Clocking provided by DCE )
STARTUP                         ( Configure software )
```

9.5 Frame Layer

IDACOM's X.25 Emulation implements an automatic layer 2 state machine. Refer to Section 12 for additional information.

```
                    Frame Layer Menu

   Emulation            AUTOMATIC    T1 Timer (Sec)     3.0
 → Modulo Detection     AUTOMATIC    Idle Timer (Sec)   1.0
   Max Tx Frame Size    128          N2 Retry Counter   10
   Max Rx Frame Size    128          Window Size        2
   Sequence Numbering   MOD 8        Initial Poll       P=1
```

Figure 9–2 Frame Layer Menu

→ *Emulation*

L2_ON (--)

Activates the layer 2 state machine (default) resulting in automatic responses to all received frames.

AUTOMATIC function key

L2_OFF (--)

Deactivates the layer 2 state machine resulting in no automatic responses to all received frames.

MANUAL function key

→ *Modulo Detection*

AUTO–MOD (-- address)

Contains the current setting of the modulo (sequence numbering) detection mechanism when a SABM or SABME is received. Valid values are 1 for automatic (default) and 0 for manual.

→ *Max Tx Frame Size*
Specifies the maximum number of bytes in transmitted frames.

=FRAME_SIZE (frame size --)
Specifies the maximum frame size for transmitted frames. Valid values are 1 through 4110 bytes (default is 261).

Example:
Set the maximum frame size to 517.
```
517 =FRAME_SIZE
```

→ *Max Rx Frame Size*
Specifies the maximum number of bytes in received frames. If the emulation is configured as DTE, this acts as the DCE N1 system parameter; otherwise, it is the DTE N1 system parameter. This frame size is used to check against the length of a received frame. Valid values are 1 through 4110 (default if 261).

=RX_FRAME_SIZE
Sets the maximum frame size for the received frame.

→ *Sequence Numbering*
Selects modulo 8/128 method of decoding and emulation for the frame layer.

L2_MOD8 (--)
Selects modulo 8 method of decoding and emulation (default) for the frame layer. SABM is used for link setup.

MOD 8 function key

L2_MOD128 (--)
Selects modulo 128 method of decoding and emulation for the frame layer. SABME is used for link setup.

MOD 128 function key

NOTE
When a SABM or SABME is received, the program is automatically placed into modulo 8 or 128, respectively (only if automatic modulo detection is selected).

→ *T1 Timer (Sec)*
T1-VALUE (-- address)
Contains the duration, in tenths of seconds, of the T1 timer. Valid values are 0 through 999999.9 (default is 3 seconds). When the T1 timer expires, a command frame is transmitted. This timer is not used when the value is set to 0. See the CCITT X.25 (1980/1984) Recommendations.

Example:
Set the T1 timer to 4 seconds.
```
40 T1-VALUE !
```

Modify T1 Timer function key

→ *Idle Timer (Sec)*

T1–IDLE (-- address)

Contains the duration, in tenths of seconds, of the link idle timer. Valid values are 0 through 999999.9 seconds (default is 30 seconds). When this timer expires, polling resumes to maintain activity on the link. Automatic polling does not occur when the value is set to 0.

Example 1:
Turn off automatic polling.

```
0 T1-IDLE !
```

INACTIVE function key

Example 2:
Set the link idle timer to 20 seconds.

```
200 T1-IDLE !
```

Modify Link Idle Timer function key

→ *N2 Retry Count*

Sets the frame retransmission counter. N2 specifies the maximum number of frame retransmissions following expiration of timer T1.

N2 (-- address)

Contains the RC and RCB variables. The RC and RCB variables count the maximum number of attempts to successfully send a frame (default is 10). Valid values are 1 through 9999999.

Example:
Set the N2 retry count to 5.

```
5 N2 !
```

Modify N2 Retry Count function key

RC (-- address)

Contains the recovery transmission counter used when retransmitting frames. The counter is initially set to the value in N2 and decremented each time a frame is retransmitted.

RCB (-- address)

Contains the recovery transmission counter used in remote busy state when retransmitting frames. The counter is initially set to the value in N2 and decremented each time a frame is retransmitted. RCB is used with the T1 timer to determine appropriate emulation response.

→ *Window Size*

Specifies the frame window size (maximum number of unacknowledged I frames). Valid values are 1 through 7 for modulo 8 (default is 7), and 1 through 127 for modulo 128 (default is 7). This value is stored in the K variable (see Section 11.1).

Example:
Set the frame window size to 2.

```
2 K !        ( Store the value 2 in the K variable )
```

→ *Initial Poll*

P0 (--)

Forces the poll bit to 0 in the next transmitted command frame.

P=0 function key

P1 (--)

Forces the poll bit to 1 in the next transmitted command frame.

P=1 function key

9.6 Packet Layer

IDACOM's X.25 Emulation implements an automatic layer 3 state machine (refer to Section 12 for more information). Depending on the emulation mode selected, either the DCE or DTE Packet Layer Menu is displayed. DTE emulation uses timers T20 to T23; DCE emulation uses timers T10 to T13. All other configuration commands are used by both emulation modes.

```
                         DTE Packet Layer Menu

  Packet Layer:
  → Emulation              AUTOMATIC      T20 Timer (Sec)   180.0
    Max Data Size          128            T21 Timer (Sec)   200.0
    Sequence Numbering     MOD 128        T22 Timer (Sec)   180.0
                                          T23 Timer (Sec)   180.0
```

Figure 9–3 DTE Packet Layer Menu

→ *Emulation*

L3_ON (--)

Activates the layer 3 state machine (default) resulting in automatic responses to received packets.

AUTOMATIC function key

L3_OFF (--)

Deactivates the layer 3 state machine resulting in no automatic responses to received packets.

MANUAL function key

→ *Max Data Size*
=SIZE (n --)
Specifies the maximum number of bytes in the data field of transmitted or received data packets for all logical channels. Valid values are 0 through 4100 (default is 128).

Example:
Set the maximum data field size to 512.
```
512  =SIZE
```

NOTE
The maximum frame size should be sufficiently larger than the maximum data size to allow for the address and control fields plus the data packet header.

→ *Sequence Numbering*
PACKET_MOD8 (--)
Selects modulo 8 method of sequence numbering for the packet layer on all LCN's.

MOD 8 function key

PACKET_MOD128 (--)
Selects modulo 128 method of sequence numbering (default) for the packet layer on all LCN's.

MOD 128 function key

NOTE
When the monitor detects a GFI (general format identifier) of either modulo 8 or 128, the corresponding decoding and reporting mechanism is used.

When the emulation detects a GFI that does not match that set by these commands:
- *a DTE emulation ignores the received packet; and*
- *a DCE emulation responds with a diagnostic packet indicating the reception of an invalid GFI.*

The following timers are used for DTE emulation.

→ *T20 Timer (Sec)*
T20–VALUE (-- address)
Contains the duration, in tenths of seconds, of the T20 timer (default is 180 seconds). The T20 timer is started when the emulation is a DTE and a restart request is transmitted using T20-VALUE to set timeout. This timer is not used when the value is set to 0. For appropriate action by the DTE when this timer expires, see TABLE D-2/X.25 in the CCITT X.25 (1984) Recommendation.

Example:
Set the T20 timer to 240 seconds.
```
2400 T20-VALUE !
```

Modify T20 Restart Timer function key

→ *T21 Timer (Sec)*

T21–VALUE (-- address)

Contains the duration, in tenths of seconds, of the T21 timer (default is 200 seconds). The T21 timer is started when the emulation is a DTE and a call request is transmitted using T21–VALUE to set timeout. This timer is not used when the value is set to 0. For appropriate action by the DTE when this timer expires, see TABLE D–2/X.25 in the CCITT X.25 (1984) Recommendation.

Example:
Set the T21 timer to 240 seconds.

```
2400 T21-VALUE !
```

Modify T21 Call Timer function key

→ *T22 Timer (Sec)*

T22–VALUE (-- address)

Contains the duration, in tenths of seconds, of the T22 timer (default is 180 seconds). The T22 timer is started when the emulation is a DTE and a reset request is transmitted using T22–VALUE. This timer is not used when the value is set to 0. For appropriate action by the DTE when this timer expires, see TABLE D–2/X.25 in the CCITT X.25 (1984) Recommendation.

Example:
Set the T22 timer to 1 minute.

```
600 T22-VALUE !
```

Modify T22 Reset Timer function key

→ *T23 Timer (Sec)*

T23–VALUE (-- address)

Contains the duration, in tenths of seconds, of the T23 timer (default is 180 seconds). The T23 timer is started when the emulation is a DTE and a clear request is transmitted using T23–VALUE. This timer is not used when the value is set to 0. For appropriate action by the DTE when this timer expires, see TABLE D–2/X.25 in the CCITT X.25 (1984) Recommendation.

Example:
Set T23 timer to 4 minutes.

```
2400 T23-VALUE !
```

Modify T23 Clear Timer function key

The following timers are used for DCE emulation.

```
                        DCE Packet Layer Menu

Packet Layer:
→ Emulation            AUTOMATIC      T10 Timer (Sec)    60.0
  Max Data Size        128            T11 Timer (Sec)    180.0
  Sequence Numbering   MOD 128        T12 Timer (Sec)    60.0
                                      T13 Timer (Sec)    60.0
```

Figure 9–4 DCE Packet Layer Menu

→ *T10 Timer (Sec)*
T10–VALUE (-- address)
Contains the duration, in tenths of seconds, of the T10 timer (default is 60 seconds). The T10 timer is started when the emulation is a DCE and a restart indication is transmitted using T10-VALUE to set timeout. This timer is not used when the value is set to 0. For appropriate action by the DCE when this timer expires, see TABLE D-1/X.25 in the CCITT X.25 (1984) Recommendation.

Example:
Set the T10 timer to 2 minutes.

```
1200 T10-VALUE !
```

Modify T10 Restart Timer function key

→ *T11 Timer (Sec)*
T11–VALUE (-- address)
Contains the duration, in tenths of seconds, of the T11 timer (default is 180 seconds). The T11 timer is started when the emulation is a DCE and an incoming call is transmitted using T11-VALUE to set the timeout. This timer is not used when the value is set to 0. For appropriate action by the DCE when this timer expires, see TABLE D-1/X .25 in the CCITT X.25 (1984) Recommendation.

Example:
Set the T11 timer to 200 seconds.

```
2000 T11-VALUE !
```

Modify T11 Call Timer function key

→ *T12 Timer (Sec)*
T12–VALUE (-- address)
Contains the duration, in tenths of seconds, of the T12 timer (default is 60 seconds). The T12 timer is started when the emulation is a DCE and a reset indication is transmitted using T12-VALUE to set timeout. This timer is not used when the value is set to 0. For appropriate action by the DCE when this timer expires, see TABLE D-1/X.25 in the CCITT X.25 (1984) Recommendation.

Example:
Set the T12 timer to 30 seconds.

```
300 T12-VALUE !
```

Modify T12 Reset Timer function key

→ *T13 Timer (Sec)*
T13–VALUE (-- address)
Contains the duration, in tenths of seconds, of the T13 timer (default is 60 seconds). The T13 timer is started when the emulation is a DCE and a clear indication is transmitted using T13-VALUE to set timeout. This timer is not used when the value is set to 0. For appropriate action by the DCE when this timer expires, see TABLE D-1/X.25 in the CCITT X.25 (1984) Recommendation.

Example:
Set the T13 timer to 90 seconds.

```
900 T13-VALUE !
```

Modify T13 Clear Timer function key

9.7 Facilities

```
                         Facility Menu

   → Call Request Facility     NONE
     User Defined Facility     ---

     Call Accept/Connect       USE ADDRESS
     Call Accept Facility      ECHO
     User Defined Facility     ---

     Call User Data            NONE
     Clear User Data           NONE
```

Figure 9–5 Facility Menu

SET_FAC_LEN (n --)

Specifies the maximum facility length of the received packet. Valid values are 0 through 63 for the 1980 Recommendation, and 0 through 109 for the 1984 Recommendation.

→ *Call Request Facility*

NO_FAC (--)

No facility data is included on any of the 255 channels when transmitting call request/incoming call packets (default).

NONE function key

YES_FAC (--)

Automatically negotiates data packet size, packet window, throughput class, and fast select on all 255 channels when transmitting call request/incoming call packets.

NEGOTIATE function key

USER_FAC (--)

Enables facility negotiation on all 255 channels. Facility fields defined with the MAKE_FAC command are transmitted within call request/incoming call packets.

USER DEFINED function key

=CLASS (throughput class --)

Specifies the throughput facility class on all 255 channels. This facility is used in a call request/incoming call facility negotiation. It expects a numerical value as an input parameter and has no output parameters. Valid values are 75, 150, 300, 600, 1200, 2400, 4800, 9600 (default), 19200, and 48000.

Example:
Set the throughput facility class to 2400.

```
2400  =CLASS
```

→ *User Defined Facility*
MAKE_FAC (string --)
Specifies the facility field used in transmitted call request/incoming call packets when USER_FAC is called. The first byte in the string is the facility length, with following bytes used as facilities. These facilities should be defined in hex byte format as shown in the example. The maximum length of the facility field is 109 octets.

Example:
Define a facility for a packet size negotiation of 256.

```
X" 03420808"  MAKE_FAC
```

The first byte in the hex string (03) indicates that the following facility field has a length of 3 bytes.

The second byte (42) indicates that the facility being negotiated is the packet size.

Bytes 3 and 4 (0808) indicate that the packet size is 256 for both the called DTE and calling DTE.

Modify Facility function key

→ *Call Accept/Connect*
YES_CA (--)
Uses the address field (default), received in call request/incoming call packets, as the address field in transmitted call accept/connect packets on all 255 channels.

USE ADDRESS function key

NO_CA (--)
No address, facility, or call user data fields are used in transmitted call accept/connect packets.

NO ADDRESS function key

→ *Call Accept Facility*
NO_CAFAC (--)
The facility field in call accept/connect packets is not used.

NONE function key

ECHO_CAFAC (--)
Echoes the facility field (default) received in call request/incoming call packets in the call accept/connect packet on all 255 channels.

ECHO function key

USER_CAFAC (--)
Enables facility negotiation on all 255 channels. Facility fields defined with the MAKE_CAFAC command are transmitted within call accept/connect packets.

USER DEFINED function key

→ *User Defined Facility*
MAKE_CAFAC (string --)
Specifies the facility field used in transmitted call accept/connect packets when USER_CAFAC is called. The first byte in the string is the facility length, with following bytes to be used as facilities. These facilities should be defined in hex byte format as shown in the example. The maximum length of the facility field is 63 octets for the 1980 Recommendation, and 109 octets for the 1984 Recommendation.

Example:
Define a facility for a packet size negotiation of 256 in call accept packets.
```
X" 03420808" MAKE_CAFAC
```

The first byte in the hex string (03) indicates that the following facility field has a length of three bytes.

The second byte (42) indicates that the facility being negotiated is packet size.

Bytes 3 and 4 (0808) indicate that the packet size is 256 for both the called and calling DTE.

Modify Facility function key

MAKE_RFAC (string --)
Specifies the registration field in transmitted registration request/confirmation packets on channel 0 (maximum length is 109 bytes).

Example:
Define a registration for a packet size negotiation of 256 in call accept packets.
```
X" 03420808" MAKE_RFAC
```

The first byte in the hex string (03) indicates that the following registration field has a length of three bytes.

The second byte (42) indicates that the facility being negotiated is packet size.

Bytes 3 and 4 (0808) indicate that the packet size is 256 for both the called and calling DTE.

Modify Registration function key

→ *Call User Data*
MAKE_CUD (string --)
Specifies the call user data field used in transmitted call request/incoming call and call accept/connect packets on all 255 channels (maximum length is 64 bytes). This field is also used in clear request packets when extended format is used.

Example 1:
Define a call user data field that contains 11 characters.
```
X" C0000000030100258000064" MAKE_CUD
```

Example 2:
Clear the call user data field.
```
NO MAKE_CUD
```

Modify Call User Data function key

→ *Clear User Data*
MAKE_CLRUD (string --)
Specifies the clear user data field used in transmitted clear request indication packets on all 255 channels (maximum length is 64 bytes).

Example 1:
Define a clear user data field that contains 11 characters.
```
X" C0000000030100258000064" MAKE_CLRUD
```

Example 2:
Clear the clear user data field.
```
NO MAKE_CLRUD
```

Modify Clear User Data function key

9.8 LCN Setup

The X.25 Emulation supports 255 logical channels which can be set to any of 4096 LCN's (logical channel numbers). Logical channels are assigned and configured for SVC/PVC, called address, calling address, packet window size, and data echo from LCN Setup Menu 1.

LCN Setup Menu 1

	LCN	TYPE	Called Address	Calling Address	Window	Echo
→ CH1	1	SVC	43042001	33001001	2	OFF
CH2	2	SVC	43042002	33001002	2	OFF
CH3	3	SVC	43042003	33001003	2	OFF
CH4	4	SVC	43042004	33001004	2	OFF
CH5	5	SVC	43042005	33001005	2	OFF
CH6	6	SVC	43042006	33001006	2	OFF
CH7	7	SVC	43042007	33001007	2	OFF
CH8	8	SVC	43042008	33001008	2	OFF
CH9	9	SVC	43042009	33001009	2	OFF
CH10	10	SVC	43042010	33001010	2	OFF

Figure 9–6 LCN Setup Menu 1

CH (logical channel --)
Specifies the logical channel to use as the current channel. Valid values are 1 through 255.

CH1 (--)
Uses logical channel 1 as the current channel. The CH2 to CH255 commands function in the same manner for selecting the 255 available channels.

The LCN of the currently selected logical channel can also be changed using the =LCN command.

NOTE
If more than one channel has the same LCN value, the automatic emulation uses the first one found on the LCN Setup Menus. Errors in protocol can be forced by sending out packets on a second channel with the same LCN by using the CH1 to CH255 commands.

=LCN (LCN number --)
Specifies the LCN of the currently selected logical channel. Valid values are 0 through 4095.

Example:
Set the LCN on channel 30 to 225.
```
CH30  225  =LCN
```

Modify LCN function key

SVC (--)

Sets the currently selected channel as an SVC (switched virtual circuit).

Example:
Set channel 3 as SVC.

```
CH3 SVC
```

SVC function key

PVC (--)

Sets the currently selected channel as a PVC (permanent virtual circuit).

Example:
Set channel 2 as PVC.

```
CH2 PVC
```

PVC function key

The called and calling addresses can be modified using the =CALLED and =CALLING commands. The corresponding address fields can be cleared using the LCNCALLED and LCNCALLING strings.

=CALLED (string --)

Specifies the called address field for subsequent call request packets on the currently selected logical channel. Up to 15 decimal digits can be specified within the ASCII string.

Example:
Define a called address of 1234567890 on channel 2.

```
CH2  " 1234567890"  =CALLED
```

Modify Called function key

LCNCALLED (-- address)

Contains a 16 byte string identifying the called address field defined with the =CALLED command. The first byte of this string contains the length of the called digits.

In the previous example, where the called address is defined as 1234567890, LCNCALLED contains the following hex values:

0A31323334353637383930

The first byte is the length of the defined called address (hex 0A, decimal 10). The remaining bytes are the hex values for ASCII representation of decimal 1234567890.

Example:
Clear the called address field on channel 3 (by setting the length to 0).

```
CH3  0 LCNCALLED  !
```

Modify Called function key

=CALLING (string --)

Specifies the calling address field for subsequent call request packets on the currently selected logical channel. Up to 15 decimal digits can be specified within the ASCII string.

Example:
Define a calling address of 1234567890 on channel 2.

```
CH2  " 1234567890"  =CALLING
```

Modify Calling function key

LCNCALLING (-- address)

Contains a 16 byte string identifying the calling address field defined with the =CALLING command. In the previous example, where the calling address is defined as 1234567890, LCNCALLING contains the following hex values: 0A31323334353637383930.

The first byte of the length is the defined calling address (hex 0A, decimal 10). The remaining bytes are the hex values for ASCII representation of decimal 1234567890.

Example:
Clear the calling address field on channel 3.

```
CH3  0  LCNCALLING  !
```

Modify Calling function key

=WINDOW (window size --)

Specifies the maximum packet window size of the currently selected logical channel. Valid values are 1 through 7 for modulo 8, and 1 through 127 for modulo 128 (default is 2).

Example:
Set the maximum packet window size to 7 for logical channel 10.

```
CH10 7  =WINDOW
```

Modify Window function key

ECHO_ON (--)

Echoes the received data packet on the currently selected channel.

Example:
Echo all data packets on CH55.

```
CH55  ECHO_ON
```

ECHO ON function key

ECHO_OFF (--)

Disables the data packet echo (default) on the currently selected channel.

Example:
Turn off the data packet echo enabled in the previous example.

```
CH55   ECHO_OFF
```

ECHO OFF function key

Each of the 255 logical channels can be configured for fast select facility, clear request format, and clear confirm format from the LCN Setup Menu 2.

LCN Setup Menu 2

	LCN	Fast Select	Clear Request	Clear Confirm
→ CH1	1	OFF	Not Extended	Not Extended
CH2	2	OFF	Not Extended	Not Extended
CH3	3	OFF	Not Extended	Not Extended
CH4	4	OFF	Not Extended	Not Extended
CH5	5	OFF	Not Extended	Not Extended
CH6	6	OFF	Not Extended	Not Extended
CH7	7	OFF	Not Extended	Not Extended
CH8	8	OFF	Not Extended	Not Extended
CH9	9	OFF	Not Extended	Not Extended
CH10	10	OFF	Not Extended	Not Extended

Figure 9–7 LCN Setup Menu 2

The following commands define the use and format of facility fields.

FAST_SELECT_OFF (--)

Disables the fast select facility (default) on the currently selected logical channel.

Example:
Turn off the fast select facility on channel 2.

```
CH2   FAST_SELECT_OFF
```

FAST SELECT OFF function key

FAST_SELECT_ON (--)

Enables the fast select facility with no restrictions in call request packets on the currently selected logical channel.

Example:
Turn on the fast select facility for channel 2.

```
CH2  FAST_SELECT_ON
```

ON function key

FAST_SELECT_RESTRICTION (--)

Enables the fast select facility with restriction on the currently selected logical channel. When the FAST_SELECT_RESTRICTION is active and a call request packet is received, the emulation transmits a clear request packet which contains address, facility, and user data fields.

Example:
Activate the fast select facility for channel 3.

```
CH3  FAST_SELECT_RESTRICTION
```

WITH RESTRICTION function key

Clear request packets contain calling and called address fields, facility fields, and optional clear user data in fast select mode if the CLEARREQ_EXT command is issued.

CLEARREQ_NOT_EXT (--)

Extended format in the transmitted clear request packets is not used on the currently selected logical channel. This is the default mode for all logical channels and applies to both DCE and DTE ends.

Example:
Turn off extended format in clear request packets on channel 1 for transmission.

```
CH1  CLEARREQ_NOT_EXT
```

NOT EXTENDED function key

CLEARREQ_EXT (--)

Uses extended format in the transmitted clear request packets on the currently selected logical channel. When this command is issued, the diagnostic code field, address length fields, and facility length fields must be present in the clear request packet. The clear user data field is optional.

CLEARREQ_EXT is selected per channel and applies to both DTE and DCE ends. It remains in effect until CLEARREQ_NOT_EXT is issued.

Example:
Use extended format in clear request packets on channel 4.

```
CH4  CLEARREQ_EXT
```

CLEAR REQUEST EXTENDED function key

NOTE
If the emulation partner is also an IDACOM tester, this command should also be issued on the partner for the same logical channel.

Clear confirmation packets contain calling and called address fields and facility fields if the CLEARCONF_EXT command is issued.

CLEARCONF_NOT_EXT (--)

Extended format in the transmitted clear confirmation packets is not used on the currently selected logical channel. This is the default mode for all logical channels and applies to the DCE end.

Example:
Turn off extended format in clear confirmation packets on channel 1.

```
CH1  CLEARCONF_NOT_EXT
```

NOT EXTENDED function key

CLEARCONF_EXT (--)

Uses extended format in the transmitted clear confirmation packets on the currently selected logical channel. When CLEARCONF_EXT is issued, the diagnostic code field, the address length fields, and the facility length fields must be present in the clear confirm packet.

The DTE side can receive, but not transmit, a clear confirm packet in extended format. The DCE side can receive or send clear confirm packets in extended format. CLEARCONF_EXT remains in effect until CLEARCONF_NOT_EXT is issued.

Example:
Use extended format in clear confirmation packets on channel 4.

```
CH4  CLEARCONF_EXT
```

CLEAR CONFIRM EXTENDED function key

NOTE
If the emulation partner is also an IDACOM tester, CLEARCONF_EXT should also be issued on the partner for the same logical channel.

10
EMULATION ARCHITECTURE

This section describes the structure of the X.25 Emulation. The IDACOM X.25 Emulation program is a combination of the complete X.25 Monitor application package together with an emulation state machine. All commands available in the X.25 Monitor are also available in the X.25 Emulation. The program's data flow is detailed for the reception of protocol events (frames, lead changes, or timers) and generating responses to these events.

Received frames are first decoded in the emulation decode block and then passed on to the test manager if a test script is running. The test manager can generate data, start timers, etc., in response to the received frame, or strictly recognize that a particular frame has been received with no associated output. After the test manager has processed the received frame, the X.25 state machine processes the received frame and generates any necessary protocol responses. By handling the received frame in this sequence, the automatic state machine operation can be turned off via the test manager, prior to running the X.25 state machine.

10.1 Live Data

The X.25 Monitor decodes any received/transmitted frames and displays them for user interpretation while the X.25 state machine interprets the received frames and forces some action (usually transmitting a frame, RR, RNR, etc.).

The emulation receives events from the interface and processes them as shown in Figure 10–1.

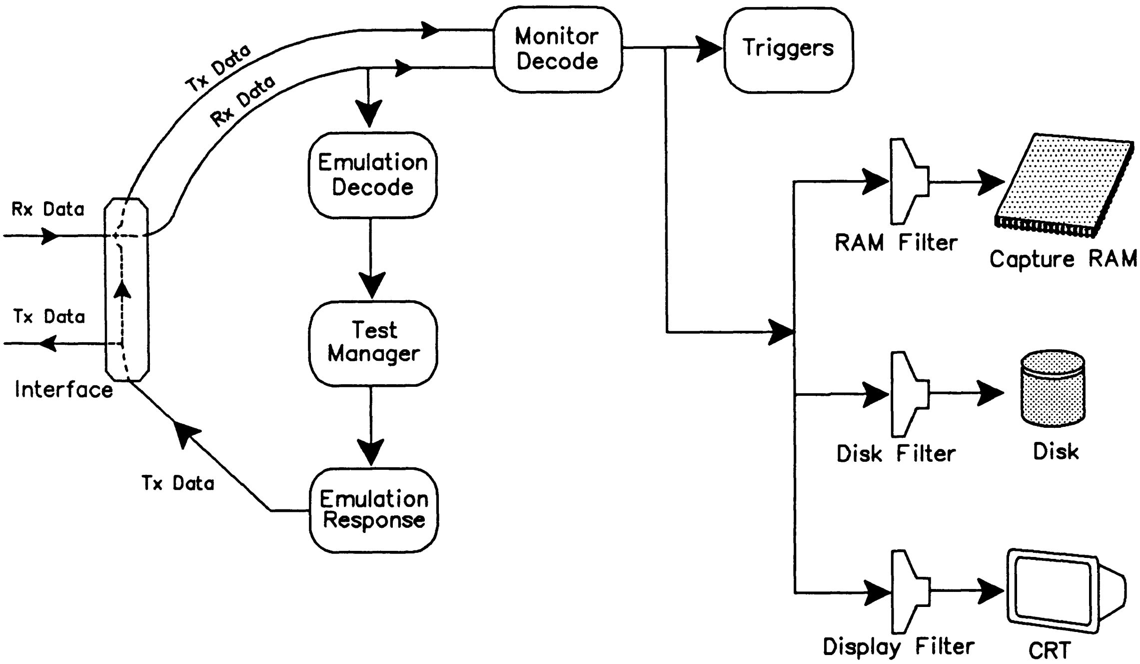


Figure 10–1 X.25 Emulation Data Flow Diagram – Live Data

By default, the X.25 Emulation captures the received/transmitted data in the capture RAM buffer and displays it on the screen in a short format report. The X.25 Emulation is running (active) and responds to all layer 2 frames and layer 3 packets. The emulation can be enabled or disabled using the EMUL_ON or EMUL_OFF commands.

Display topic
Live Data function key

MONITOR (--)
Selects the live data display mode of operation. All incoming events and transmitted frames are decoded and displayed in real–time.

EMUL_ON (--)
Enables the automatic emulation; the X.25 state machine responds to all incoming frames according to the X.25 protocol.

Emulation topic
Run Emulation function key (highlighted)

EMUL_OFF (--)
Disables the automatic emulation; the X.25 state machine does not transmit any frames without manual or test program intervention.

Emulation topic
Run Emulation function key (not highlighted)

10.2 Playback

Data can be played back from either capture RAM or disk without interfering with an active test (i.e. dropping the link) as shown in Figure 10-2.

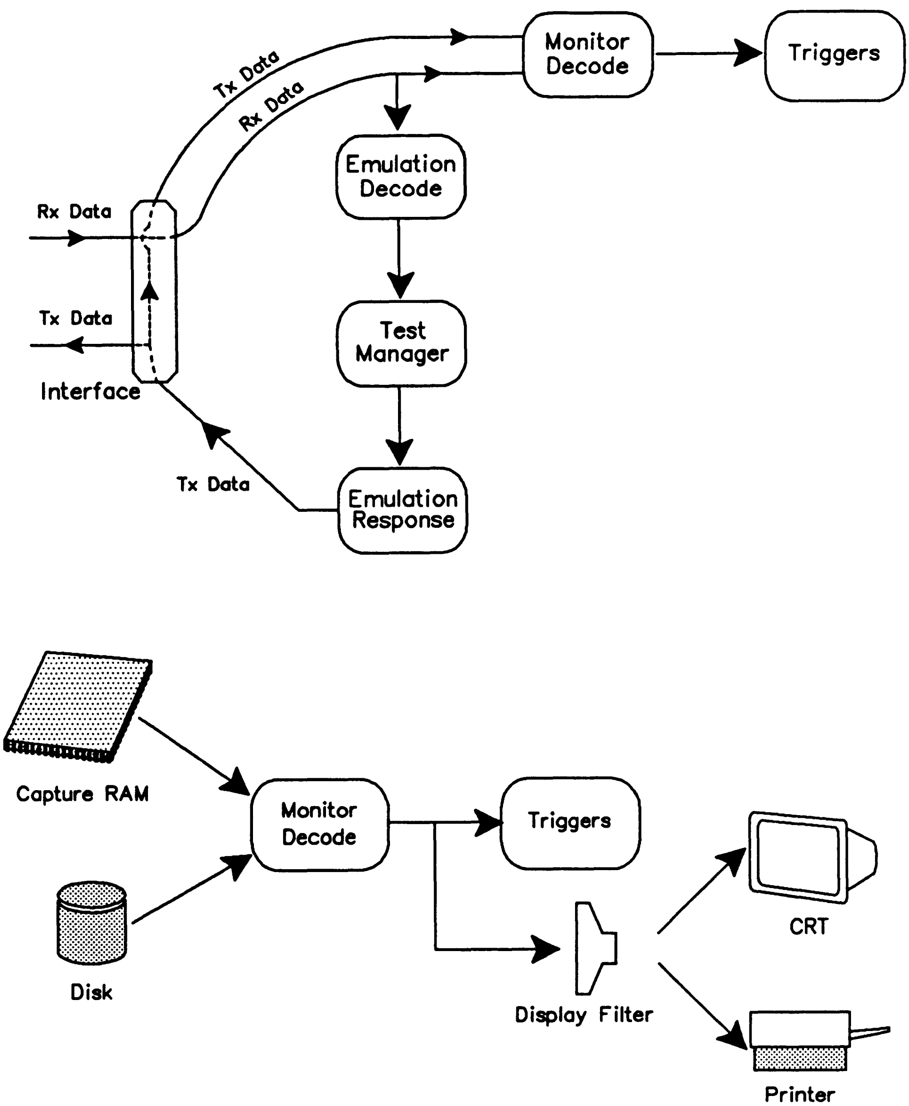


Figure 10–2 X.25 Emulation Data Flow Diagram – Offline Processing

FROM_CAPT HALT
Display topic
Playback RAM function key

FROM_DISK HALT PLAYBACK
Display topic
Playback Disk function key

HALT (--)
Selects the playback mode of operation. Data is retrieved from capture RAM or a disk file, decoded, and then displayed or printed. Capture to RAM is suspended in this mode.

10.3 Simultaneous Live Data and Playback

Live data can be recorded to disk while playing back data from capture RAM.

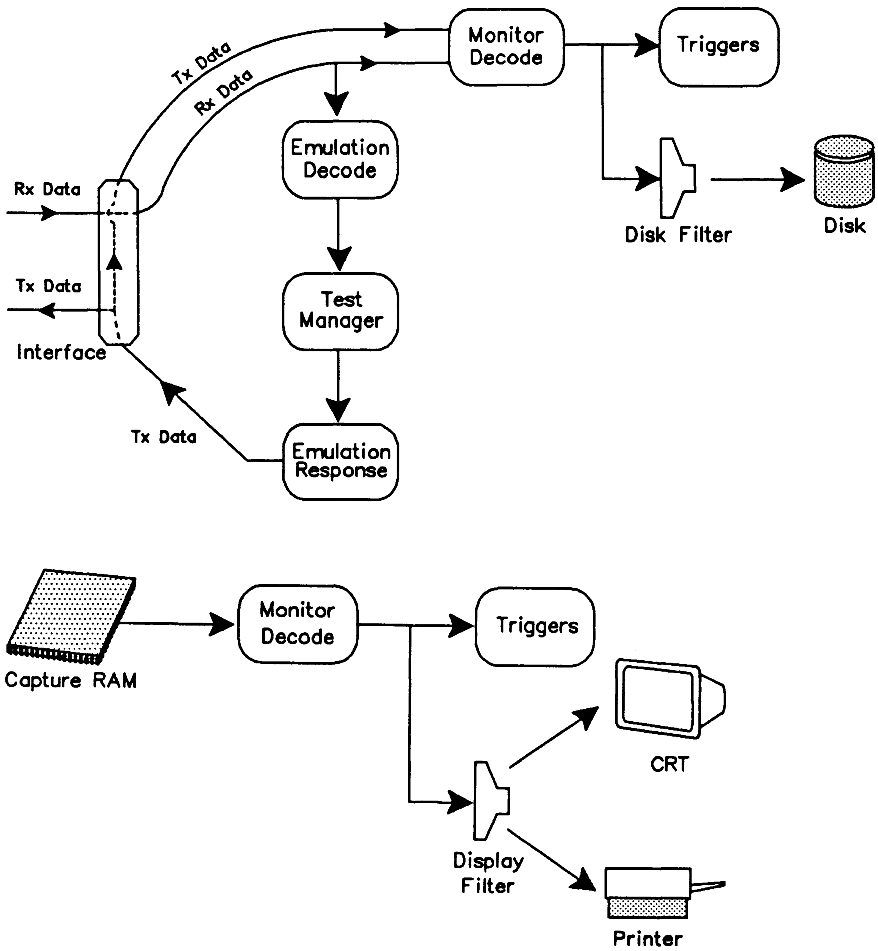


Figure 10–3 X.25 Emulation Data Flow Diagram – Freeze Mode

FROM_CAPT FREEZE
Capture topic
Record to Disk function key
Display topic
Playback RAM function key

FREEZE (--)
Enables data to be recorded to disk while data from capture RAM is played back.

11
EMULATION DECODE

This section describes the data flow diagram for the emulation decode and lists the variables in which decoded information is saved.

The X.25 Emulation operation follows the CCITT X.25 (1980/1984) Recommendations.

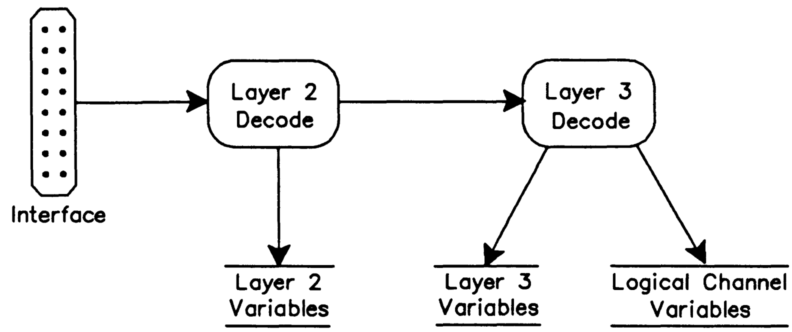


Figure 11–1 X.25 Emulation Data Flow Diagram – Decode

11.1 Communication Variables

This section describes both receive and transmit communication variables. Receive variables are set during the decode process, and contain protocol specific information as defined in the CCITT X.25 (1980/1984) Recommendations. The emulation uses the information in these variables to determine the appropriate action to external events.

Transmit variables are used when transmitting a frame and can be used in test scripts to modify the emulation response.

NOTE
These variables can be read using the @ (fetch) operation.

Layer 2

ADDRESS–A (-- address)
Contains the address (first) byte of received command frames from the DCE, or response frames from the DTE. Valid values are 3 for SLP, and 15 for MLP. This value is used by the emulation to classify incoming frames.

ADDRESS-B (-- address)
Contains the address (first) byte of received command frames from the DTE, or response frames from the DCE. Valid values are 1 for SLP, and 7 for MLP. This value is used by the emulation to classify incoming frames.

ADDR (-- address)
Contains the address (first) byte of the received frame. For received commands from the DCE or responses from the DTE, valid values are 3 for SLP, and 15 for MLP. For received commands from the DTE or responses from the DCE, valid values are 1 for SLP, and 7 for MLP.

MADDR (-- address)
Contains the address byte used in transmitting response frames. For DTE emulation, valid values are 3 for SLP, and 15 for MLP. For DCE emulation, valid values are 1 for SLP, and 7 for MLP.

YADDR (-- address)
Contains the address byte used in transmitting command frames. For DTE emulation, valid values are 1 for SLP, and 7 for MLP. For DCE emulation, valid values are 3 for SLP, and 15 for MLP.

CONTROL (-- address)
Contains the control (second) byte of the last received frame. See Tables A-3 and A-4 for possible values.

C/R (-- address)
Contains the last received frame command response indicator. The value is 0 if the last received frame was a command and 1 if a response.

FRMRW (-- address)
Contains the W bit within a transmitted FRMR frame. The first bit of the fifth byte for modulo 8, and the first bit of the seventh byte for modulo 128. The value is 1 when the control field in a received frame is undefined or not implemented.

FRMRX (-- address)
Contains the X bit within a transmitted FRMR frame with a value of 0 or 1. The second bit of the fifth byte for modulo 8, and the second bit of the seventh byte for modulo 128. The value is 1 when the control field in a received frame contains an illegal information field or when an unnumbered frame of incorrect length is received. When set to 1, the W bit must also equal 1.

FRMRY (-- address)
Contains the Y bit within a transmitted FRMR frame with a value of 0 or 1. The third bit of the fifth byte for modulo 8, and the third bit of the seventh byte for modulo 128. The value is 1 when the information field in a received I frame is longer than the established maximum length.

FRMRZ (-- address)
Contains the Z bit within a transmitted FRMR frame with a value of 0 or 1. The fourth bit of the fifth byte for modulo 8, and the fourth bit of the seventh byte for modulo 128. The value is 1 when the control field in a received frame contains an invalid N(R).

K (-- address)
Contains the frame window size (maximum number of unacknowledged I frames). Valid values are 1 through 7 for modulo 8, and 1 through 127 for modulo 128 (default is 7).

MAX_LENGTH (-- address)
Contains the maximum frame length to be transmitted (not including the FCS bytes). Valid values are 1 through 4110 (default is 261). A value of 1 can be used to create invalid frames which the emulation partner should discard.

NR (-- address)
Contains the N(R) (receive sequence count) of the last received supervisory or information frame. Valid values are 0 through 7 for modulo 8, and 0 through 127 for modulo 128. Bits 6 to 8 of the control field in information and supervisory frames represent modulo 8, and bits 10 to 16 of the control field in information and supervisory frames represent modulo 128. The X.25 Emulation performs checking procedures with the NR, K, and VS variables to determine the appropriate action.

NS (-- address)
Contains the N(S) (send sequence count) of the last received Information frame. Valid values are 0 through 7 for modulo 8, and 0 through 127 for modulo 128. Bits 2 to 4 of the control field in information frames represent modulo 8, and bits 2 to 8 of the control field in information frames represent modulo 128. The X.25 Emulation checks this value against that in the VR variable to determine the appropriate action.

P/F (-- address)
Contains the poll/final bit of the last received frame (0 or 1). Bit 5 of the control field. If the emulation receives a command frame with the P bit set to 1, the F bit is set to 1 in the next transmitted response frame it transmits.

PF (-- address)
Contains the value of the P bit used for transmitted command frames (0 or 1). It is also the expected value in P/F in the next received response frame.

STATE_L2 (-- address)
Contains a value which represents one of the layer 2 states in the layer 2 emulation (see Table 12–1). Valid values are 1 through 12.

VR (-- address)
Contains the current V(R) (receive sequence number) used when an information or supervisory frame is transmitted. Valid values are 0 through 7 for modulo 8, and 0 through 127 for modulo 128. VR is set to 0 upon reception of a UA after transmission of a SABM/SABME and incremented by one every time an error free in sequence I frame is received containing an N(S) equal to VR.

VS (-- address)
Contains the current V(S) (send sequence number) used when an information frame is transmitted. VS is set to 0 on the reception of a UA after transmission of a SABM/SABME and incremented by one every time an I frame is transmitted. Valid values are 0 through 7 for modulo 8, and 0 through 127 for modulo 128.

VSU (-- address)
Contains the V(S) (sequence number) of the lowest unacknowledged I frame. It is set to 0 when a UA is received after transmission of a SABM/SABME. When a I frame or supervisory frame is received, the current V(S) is compared to the received N(R). If the current V(S) is greater or equal to the received N(R), VSU is made equal to the N(R).

VX (-- address)
Contains the timer recovery condition. If the T1 timer runs out while waiting for acknowledgement of a transmitted I frame, the I frame is retransmitted with the poll bit set to 1. After retransmission, VX is set to the value of V(S).

Layer 3

PR (-- address)
Contains the P(R) (packet receive sequence number) of the last received data, RR, RNR, or REJ packet. Valid values are 0 through 7 for modulo 8, and 0 through 127 for modulo 128. Bits 6 to 8 of the packet identifier octet in data, RR, RNR, or REJ packets represent modulo 8. Bits 2 to 8 of octet 4 represent modulo 128.

PS (-- address)
Contains the P(S) (packet send sequence number) of the last received data packet. Valid values are 0 through 7 for modulo 8, and 0 through 127 for modulo 128. Bits 6 to 8 of the packet identifier octet in data packets for modulo 8. Bits 2 to 8 of octet 3 in data packets for modulo 128.

RCAUSE (-- address)
Contains the received cause byte. Octet 4 of the following packets:
- Clear request/indication
- Reset request/indication
- Restart request/indication

If the emulation is a DCE, the X.25 Emulation checks if the cause byte has a value of 0, or is between 128 and 255. See the CCITT X.25 (1984) Recommendation for defined values for specific packets.

RDIAG (-- address)
Contains the diagnostic byte of the received packet. Valid values are 0 through 255. Octet 5 of the following packets:
- Clear request/indication
- Reset request/indication
- Restart request/indication

NOTE
See the CCITT X.25 (1980/1984) Recommendation for defined values.

REC_PKT_ID (-- address)
Contains the received packet identifier byte. Used by the emulation to determine if this is an appropriate packet to receive for current LCN state. See Table A-6 for valid values.

RECD (-- address)
Contains the D (delivery confirmation) bit of a received data packet, call request/incoming call, or call accept/connect packet (0 or 1). Bit 7 of the first packet octet.

If this bit is set to 1 in any other packet type, the decode operation indicates an invalid GFI. If the emulation is a DCE, it responds with a diagnostic packet with a value of 40 as the diagnostic code indicating an invalid GFI.

RECM (-- address)

Contains the M (more) bit of a received data packet (0 or 1). Bit 5 of octet 3 of the data packet represents modulo 8, and bit 1 of octet 4 represents modulo 128. When the M bit is set to 1, more data will follow.

RECPKTMOD (-- address)

Contains the received packet modulo. Bits 5 and 6 of the first octet of the packet. RECPKTMOD contains the hex value 10 for modulo 8, and 20 for modulo 128. Any other value is decoded as an invalid GFI. If the emulation is a DCE, it responds with a diagnostic packet with a value of 40 as the diagnostic code indicating an invalid GFI.

RECQ (-- address)

Contains the Q (qualifier) bit of a received data packet (0 or 1). Bit 8 of the first data packet octet.

If this bit is set to 1 in any other packet type, the decode operation indicates an invalid GFI. If the emulation is a DCE, it responds with a diagnostic packet with a value of 40 as the diagnostic code indicating an invalid GFI.

RIUD (-- address)

Contains the user data byte in a received interrupt packet. Octet 4 in interrupt packets. The interrupt user data field can be 1 for the 1980 Recommendation or 1 through 32 octets for the 1984 Recommendation. If the emulation is in state p1 – Ready, p2 – Calling, or p5 – Collision, the emulation responds with a clear request packet. If the emulation is in state d1 – Flow Control Ready or d8 – Remote Busy and the interrupt user data field is greater than 32 octets in length, the emulation responds with a reset request packet.

RLCN (-- address)

Contains the combined logical group identifier and logical channel number of the received packet. Valid values are 0 through 4095. Bits 1 to 4 of the first packet octet and all bits of the second packet octet.

The emulation supports up to 255 logical channels at a time. These are defined on the LCN Setup Menu 1 as CH1 through CH255. Any changes to this should be made prior to running the emulation.

DCE Emulation If RLCN contains a value other than that defined on the LCN Setup Menu 1, the emulation ignores this received packet. If in a state waiting restart confirm or link up, it responds with a diagnostic packet.

DTE Emulation If RLCN contains a value other than that defined on the LCN Setup Menu 1, the emulation ignores this received packet regardless of the state.

SCAUSE (-- address)

Contains the cause field for transmitted clear request/indication, reset request/indication, and restart request/indication packets. Used in the automatic emulation and the S:CLEARR, S:RESETR, and S:RESTARTR commands. When the automatic emulation detects an error, SCAUSE contains the following values:

Packet Type	DCE Emulation Local Procedure Error	DTE Emulation DTE Originated
Clear	19	0
Reset	5	0
Restart	1	0

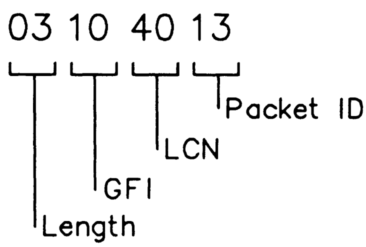
SDIAG (-- address)

Contains the diagnostic field for transmitted clear request/indication, reset request/indication, and restart request/indication packets. Used in the automatic emulation and the S:CLEARR, S:RESETR, S:RESTARTR, and S:DIAG commands. See the CCITT X.25 (1984) Recommendation, Annex E for defined values. After the packet is transmitted, SDIAG is reset to 0.

SDIAG–EXP (-- address)

Contains the diagnostic explanation field used in the S:DIAG command. The first byte contains the length of this field (up to 3 bytes). The remaining bytes contain the first 3 or less octets of the last received packet. To change this field in a transmitted packet, change the value in this variable just prior to transmission.

Example 1:
If a clear request packet is received on LCN 64, SDIAG–EXP contains the hex value:



Example 2:
Indicate that a short packet (2 bytes) was received on LCN 1.

```
0X02100100 SDIAG-EXP !
```

SENDM (-- address)

Contains the value of the M bit sent in the next data packet (default is 0.)

SIUD (-- address)

Contains the interrupt user data byte used when sending interrupt request packets. Octet 4 in interrupt request packets (default is X" 00"). Maximum values are 32 octets for CCITT X.25 (1984); 1 octet for CCITT X.25 (1980).

SET_INT_LEN (# --)
Specifies the maximum length of interrupt user data of the received frame/packet. Maximum length is 32 octets for 1984 and 1 octet for 1980. Valid values are 1 through 32 octets for the 1984 Recommendation, and 1 octet for the 1980 Recommendation.

STATE_L3 (-- address)
Contains a value which represents one of the layer 3 states in the layer 3 emulation (see Table 12-2). Valid values are 1 through 4.

SENDQB (-- address)
Contains the Q (qualifier) bit for transmitted packets (0 or 1). Bit 8 of the first packet octet. If this bit is set to 1 in a packet type other than a data packet during DTE emulation, the partner should respond with a diagnostic packet with a diagnostic code of 40, indicating an invalid GFI.

SENDDB (-- address)
Contains the D (delivery confirmation) bit of a transmitted packet (0 – default or 1). Bit 7 of the first packet octet. The emulation automatically sets it to 1 if a call request/incoming call packet is received with a D bit of 1. SENDDB is reset to 0 when the call accept/connect packet is transmitted. See RECD for expected response from the emulation partner.

SENDGFI (-- address)
Contains the GFI modulo indicator used for transmitting packets. Bits 5 and 6 of the first packet octet. The default value is hex 10 for packet modulo 8, and hex 20 for packet modulo 128.

RCALLED (-- address)
Contains a 16 byte string identifying the called address field of a received packet. It is used by the S:CALLC commands. See the example under RCALLING; string contents and handling are similar.

RCALLING (-- address)

Contains a 16 byte string identifying the calling address field of a received packet. It is used by the S:CALLC commands.

Example:
Check whether the CALLING number of the last received call request matches a predefined number (eg. 43042001). The ?MATCH command is used to determine if the calling address in a received call request packet matches the defined address.

```
TCLR

#IFNOTDEF MATCH-CALL
      0 VARIABLE MATCH-CALL 12 ALLOT
                                  ( MATCH-CALL will contain the desired calling address )
#ENDIF

0 STATE_INIT{
      " 43042001"
      COUNT                  ( Obtain # of digits in calling address )
      15 MIN                 ( Set maximum to 15 digits )
      DUP MATCH-CALL C!      ( Write # of digits to first byte of matching string )
      MATCH-CALL 1 + SWAP CMOVE      ( Write calling address in matching string )
  }STATE_INIT

0 STATE{
      R*CALLREQ 1 ?RX
      ACTION{
          RCALLING           ( Get this calling address )
          COUNT              ( Get # of digits in this calling address )
          MATCH-CALL         ( Get desired matching address )
          ?MATCH             ( Compare addresses )
          IF
              BEEP           ( Notify user )
              T." Address Match has occurred."  TCR
          ENDIF
      }ACTION
  }STATE
```

RCUD (-- address)

Contains a 128 byte string identifying the call user data of a received packet.

Example:
The tester receives a call request packet with a call user field that contains 11 characters, i.e. the hex characters C0000000003010025800064.

Obtain the length of the call user data field (in this case 11).

```
RCUD C@
```

If the call user data field is present, its use and format are determined by bits 7 and 8 of the first octet. This octet can be obtained by RCUD 1 + C@. Refer to CCITT the X.244 (1984) Recommendation for further information on call user data.

RFAC (-- address)

Contains a 256 byte string identifying the facility field of a received call request/incoming call, call accept/connect, clear request/clear indication, or clear confirmation packet.

NOTE
The maximum facility length is 109 octets. If the received length is longer than 109 octets, the X.25 Emulation transmits a clear request packet.

Example:
The tester receives a call request packet with a facility field that contains 8 characters, i.e. the hex characters 02AA420808430303.

Obtain the length of the facility field (8 in this case).
```
RFAC C@
```

Obtain the first octet of the first facility (hex 02 in this case). This indicates that the first facility is:

- a Class A facility, i.e. it is followed by a two octet parameter field; and
- throughput class negotiation.

```
RFAC 1 + C@
```

Obtain the throughput class octet (hex AA in this case). This indicates that the throughput class for the called DTE and calling DTE is 9600 bits.
```
RFAC 2 + C@
```

Obtain the first octet of the second facility (hex 42 in this case). This indicates that the second facility is:

- a Class B facility, i.e. it is followed by a two octet parameter field; and
- packet size.

```
RFAC 3 + C@
```

Obtain the packet size for the called DTE (hex 08 in this case). This indicates a packet size of 256.
```
RFAC 4 + C@
```

Obtain the packet size for the calling DTE (hex 08 in this case). This indicates a packet size of 256.
```
RFAC 5 + C@
```

Obtain the first octet of the third facility (hex 43 in this case). This indicates that the third facility is:

- a Class B facility, i.e. it is followed by a two octet parameter field; and
- window size.

```
RFAC 6 + C@
```

Obtain the window size of the called DTE (hex 03 in this case). This indicates a window size of 3.
```
RFAC 7 + C@
```

Obtain the window size of the calling DTE (hex 03 in this case). This indicates a window size of 3.
```
RFAC 8 + C@
```

NOTE
Refer to the CCITT X.25 (1984) Recommendation, Section 7 Formats for Facility Fields and Registration Fields for further information on decoding facilities.

SET_FAC_LEN (length --)
Sets the maximum length of the facility field in received call/clear packets. Default value is 63 octets for the 1980 Recommendation, and 109 octets for the 1984 Recommendation.

Logical Channel Variables

The MLP emulation supports simultaneous execution of 255 logical channels.

The logical channel variables access parameters from the currently selected logical channel. The default is CH1 as defined on LCN Setup Menu 1. The current logical channel can be set using the following methods:

- Under the **L3Send** topic, press the *Enter LCN* function key. If the entered value (1 through 4095) matches one of those defined on LCN Setup Menu 1, the value in the LCN variable will be set to that entered value.
- Use one of the CH1 to CH255 commands (see Section 9.8).
- Use the =LCN command (see Section 9.8).

LCN (-- address)
Contains a pointer to the value of the current logical channel and logical group. Valid values are 0 through 4095 (default is 1).

LCNSTATE (-- address)
Contains a pointer to the current LCN state with one of the following values:
state p1 – Ready, state p2 – Calling, state p5 – Collision, state d1 – Flow Control Ready, state d2 – Reset, state p6 – Clearing, state d8 – Remote Busy (see Section 12.1).

LCNWINDOW (-- address)
Contains a pointer to the window size for the current logical channel. Valid values are 1 through 7 for modulo 8, and 1 through 127 for modulo 128 (default is 2).

LCNSIZE (-- address)
Contains a pointer to the size of the data field used with the DATA command. Valid values are 0 through 4110 bytes (default is 128).

LCNDSIZE (-- address)
Contains a pointer to the maximum size of the data field in data packets for the current logical channel. This is negotiated in the facility field during call setup. The standard negotiated value is 128 octets; others are 16, 32, 64, 256, 512, 1024, 2048, and 4096 octets.

NVS (-- address)
Contains a pointer to the current LCN P(S) (send sequence number) used when a data packet is transmitted. It is set to 0 when the logical channel has just entered state d1 – Flow Control Ready and incremented by one every time a data packet is transmitted. Valid values are 0 through 7 for modulo 8, and 0 through 127 for modulo 128.

NVR (-- address)

Contains a pointer to the current LCN P(R) (receive sequence number). This is the next P(S) expected to be received. It is set to 0 when the logical channel has just entered state d1 – Flow Control Ready. When a data packet is received with a valid P(S) and P(R), NVR is incremented by one. Valid values are 0 through 7 for modulo 8, and 0 through 127 for modulo 128. NVR is used when transmitting data, RR, RNR, and REJ packets.

NSU (-- address)

Contains a pointer to the LCN P(S) (sequence number) of the lowest unacknowledged data packet. It is set to 0 when the logical channel has just entered state d1 – Flow Control Ready. When a data packet is received with a valid P(S) and P(R) or an RR, RNR, or REJ packet is received with a valid P(R), the current received P(R) is written in NSU. Valid values are 0 through 7 for modulo 8, and 0 through 127 for modulo 128.

12
EMULATION RESPONSE

The X.25 Emulation is implemented as a multi-layer, state-driven protocol emulation. There are separate program modules for each protocol layer. These modules communicate with each other to implement protocol response behavior.

The emulation has been set up to run as:

- an automatic simulation which operates precisely in accordance with the CCITT X.25 (1980/1984) Recommendations;
- a semi-automatic tester. The test manager is used to build and execute test scenarios to test responses and for generation of errors (see Section 13); and
- a manual tester. The test is controlled from the user's keyboard.

12.1 Emulation State Machines

To ensure correct protocol operation, state machines have been implemented. Based on input events (i.e. received frames or packets), transitions from one state to another are made in accordance with CCITT Recommendations.

Layer 2

NEW_L2_STATE (n --)

Sets the layer 2 state machine to a specific state. Valid values are 1 through 12. The state value is stored in the STATE_L2 variable.

Example:
Put the layer 2 state machine in the disconnected state.

```
1 NEW_L2_STATE
```

State Number	State Name	Description
1	Disconnected	DTE is logically disconnected from link
2	Waiting For Disconnect	Waiting to enter state 1
3	Waiting For Operational	Waiting to enter state 6
4	Frame Rejection	In frame rejection condition
5	Busy	In local busy condition
6	Operation (Information Transfer)	Operational, in information transfer mode
7	Reject	In information transfer mode and has requested retransmission of I frames
8	Busy Reject	Entered busy condition after requesting retransmission of I frames
9	Busy/Remote Busy	Both local and remote busy
10	Remote Busy/Operational	Remote busy condition
11	Remote Busy/Reject	Remote is busy and retransmission of I frames requested
12	Busy/Remote Busy/Reject	Local busy, remote busy and requested retransmission of I frames

Table 12–1 Layer 2 States

Layer 3

NEW_L3_STATE (n --)

Sets the layer 3 state machine to a specific state. Valid values are 1 through 4. The state value is stored in the STATE_L3 variable.

Example:
Put the layer 3 state machine in the idle state.

```
1 NEW_L3_STATE
```

State Number	State Name	Description
1	Idle	Disconnected from layer 2
2	Waiting Operational	Waiting for link operational
3	Waiting Restart Confirm	Waiting for restart confirm
4	Link Up	Operational state

Table 12–2 Layer 3 States

Embedded in state 4 of the layer 3 state machine, is a state machine for each logical channel.

NEW_LCN_STATE (n --)

Sets the logical channel state of the currently selected LCN to a specific state. Valid values are 1, 2, 3, 4, 5, 6, and 8.

Example:
Put the logical channel state machine in state p1 – Ready.

```
1 NEW_LCN_STATE
```

NOTE
The layer 3 state machine should be in State 4 (i.e. Link Up) when using the NEW_LCN_STATE command.

State Number	State Name	Description
1	p1 – Ready	SVC LCN is ready to send or receive call
2	p2 – Calling	SVC LCN has sent call request, waiting for call connect
3	p5 – Collision	DTE and DCE simultaneously transmit a call request and incoming call on the same logical channel
4	d1 – Flow Control Ready	Flow control state: Ready for data
5	d2 – Reset	LCN has sent reset request, waiting for reset confirm
6	p6 – Clearing	SVC LCN has sent clear request, waiting for clear confirm
8	d8 – Remote Busy	Flow control ready with remote end busy

Table 12–3 Logical Channel States

NOTE
States 1, 2, 3, and 6 do not apply to PVC (Permanent Virtual Circuit).

12.2 Automatic Responses

The state machines normally handle the protocol automatically, i.e. there are automatic responses to received frames and packets.

Depending on the testing requirements, different levels of automatic response can be utilized.

Testing Requirement	Automatic Emulation	
	Layer 2	Layer 3
Layer 2+ simulation	L2_OFF	L3_OFF
Layer 3+ simulation	L2_ON	L3_OFF
Layer 4+ simulation	L2_ON	L3_ON

Table 12–4 Levels of Automatic Response

NOTE
The combination of L3_ON and L2_OFF is meaningless.

The following commands activate and deactivate the frame and packet state machines.

L2_ON (--)
Activates the layer 2 state machine. Automatic responses to received frames are generated.

L2_OFF (--)
Deactivates the layer 2 state machine. No automatic responses to received frames are generated.

L3_ON (--)
Activates the layer 3 state machine. Automatic responses to received packets are generated.

L3_OFF (--)
Deactivates the layer 3 state machine. No automatic responses to received packets are generated.

EMUL_ON (--)
Activates the layer 2 and layer 3 state machines (default). Automatic responses to received frames and packets are generated.

EMUL_OFF (--)
Deactivates the layer 2 and layer 3 state machines. No automatic responses to received frames and packets are generated.

Protocol state change reports can be displayed, captured, or recorded to disk along with X.25 data. These change reports are useful for tracing protocol or test manager operation. The following commands activate and deactivate the protocol state change reports.

STATE_ON (--)
Generates a trace report line for every protocol or test manager state change (default).

STATE_OFF (--)
Disables reporting of protocol or test manager state changes.

NOTE
See Section 8 for commands that activate/deactivate these reports from the display, capture, or disk recording.

12.3 State–Dependent Send Commands

Either function keys or commands are used to transmit frames or packets in conjunction with the automatic state machines. These commands or function keys force protocol state changes and the emulation thereby expects the correct response.

NOTE
When using these commands or function keys, the layer 2 and layer 3 protocol state machines must be activated. See Section 14 for examples.

Layer 2

SABM (--)
Transmits a SABM frame, starts the T1 timer, and puts the emulation in waiting for operational state. The emulation expects a UA response. When a UA is received, the emulation is put into information transfer state, and the T1 timer is stopped. If a UA is not received and the T1 timer expires, the emulation sends another SABM. The SABM sets the frame layer emulation to modulo 8.

SABME (--)
Transmits a SABME frame, starts the T1 timer, and puts the emulation in waiting for operational state. The emulation expects a UA response. When a UA is received, the emulation is put into information transfer state and the T1 timer is stopped. If a UA is not received and the T1 timer expires, the emulation sends another SABME. SABME sets the frame layer emulation to modulo 128.

DISC (--)
Transmits a DISC frame, starts the T1 timer, and puts the emulation in waiting for disconnect state. A UA or DM response is expected. When either the UA or DM is received, the emulation goes into the disconnected state. If a UA or DM is not received and the T1 timer expires, retransmission counter RC is checked to determine if it contains 0. If so, the emulation goes to the disconnected state. If not, RC is decremented by one, another DISC is transmitted, and the T1 timer is restarted.

DM (--)
Transmits a DM frame with the F bit equal to the last received P or F bit. The emulation is forced to the disconnected state.

FRMR (--)
Transmits an FRMR frame using the control field of the last received frame as the rejected frame control field. The emulation is forced into the frame rejected state. A SABM (modulo 8) or SABME (modulo 128) frame is expected before information transfer can continue.

NOTE
A SABM, SABME, DISC, DM, and FRMR frame can also be transmitted by pressing the corresponding function key under the L2Send topic.

SENDF (string --)
Transmits a user-defined string as an entire frame. This string is created by using the " string" or X" hex characters" commands. Up to 80 characters are allowed when directly input from the keyboard and 255 characters are allowed within test scripts.

Example:
```
X" 013F"  SENDF                ( Transmits a SABM with the P bit set )
X" 017F"  SENDF                ( Transmits a SABME with the P bit set )
```

Layer 3

The currently selected logical channel is:
- the last channel activated by the last CHn command;
- the value entered with the *Enter LCN* function key under the **L3Send** topic; or
- the last logical channel specific packet received.

CALL (--)
If the layer 3 and LCN state machines are in a state that allows a call packet to be sent, the emulation transmits a call request packet for a DTE emulation, or an incoming call packet for a DCE emulation. This packet is sent out on the currently selected LCN with the called and calling addresses as specified on the LCN Setup Menu.

The T21/T11 timer is started and the LCN state machine is set to state p2 – Calling. In response, the emulation expects a call connect packet if configured as DTE and a call accept if configured as DCE. If a correctly formatted call connect/accept packet is received, the LCN state machine goes to state d1 – Flow Control Ready and the T21/T11 is stopped. If the call connect/accept packet has an error in the facility field, a clear request packet is transmitted.

Example:
```
CALL        ( Send Call request/incoming call on current LCN )
CH2  CALL   ( Sends Call request/incoming call on channel 2 )
```

CLEAR (--)

If the layer 3 or LCN state machines are in a state that allows a clear packet to be sent, the emulation transmits a clear request packet for a DTE emulation, or a clear indication packet for a DCE emulation. This packet is sent out on the currently selected LCN.

The T23/T13 timer is started and the LCN state machine is set to state p6 – Clearing. In response, the emulation expects a clear confirm packet. If a correctly formatted clear confirm packet is received, the LCN state machine goes to state p1 – Ready and the T23/T13 is stopped.

Example:

```
CLEAR          ( Sends Clear request/indication on currently selected LCN )
CH3  CLEAR     ( Sends Clear request/indication on channel 3 )
```

DATA (--)

If the layer 3 or LCN state machines are in a state that allows a data packet to be sent and the packet layer window is open, the emulation transmits a data packet with the correct P(S) and P(R) with the maximum amount of data as specified on the Packet Layer Menu. This packet is sent out on the currently selected LCN. For an SVC (switched virtual circuit), a call must have been set up first.

The data field is filled with predefined text, and can be changed using the DEFINE_DATA command. A test program can determine if the window is open by using the WINDOW? command.

Example:

```
DATA           ( Sends Data packet on the currently selected LCN )
CH4  DATA      ( Sends Data packet on channel 4 )
```

DEFINE_DATA (filename --)

Defines the text within the data field of the data packet for the DATA command and the *DATA* function key. A file containing the desired text must first be created using the editor on the Home processor. DEFINE_DATA overwrites BUFFER 0 of the test manager (see the 'Using Buffers' section on page 13-15).

NOTE

If the file created by the user contains less than 1024 characters, the data field is padded out to 1024 characters.

Example:

Create a file with the name CUSTOM.F with the desired text.

- Press the **HOME** key.
- Move the topic bar to the **Files** topic.
- Insert a formatted floppy diskette in drive 0.
- Press the *Edit* function key.

The 'Edit script:' prompt is displayed.

- Type: DR0:CUSTOM.F.
- Press ↵ (RETURN).
- Enter desired text.
- Press the *Save* function key.
- Press the *Quit* function key.

Follow the instructions in the User Manual for loading the X.25 Emulation. Switch to the application processor which is running the program.

To use the previously created file:

- Press the **ESC** key to enter the command mode.
- Type: DR0 " CUSTOM.F" DEFINE_DATA
- Press the **ESC** key to leave the command mode.

Any data packets sent using the *DATA* function key under the **L3Send** topic will contain the text created by the user.

INTERRUPT (--)

If the layer 3 or LCN state machines are in a state that allows an interrupt packet to be sent, the emulation transmits an Interrupt packet. This packet is sent out on the currently selected LCN provided a call has been set up.

Example:

```
INTERRUPT        ( Sends Interrupt packet on currently selected LCN )
CH5  INTERRUPT   ( Sends Interrupt packet on channel 5 )
```

RESET (--)

If the layer 3 or LCN state machines are in a state that allows a reset request packet to be sent, the emulation transmits a reset request packet for a DTE emulation or a reset indication packet for a DCE emulation. This packet is sent out on the currently selected LCN.

The T22/T12 timer is started and the LCN state machine is set to state d2 – Reset. In response, the emulation expects a reset confirm packet. If a correctly formatted reset confirm packet is received, the LCN state machine goes to state d1 – Flow Control Ready and T22/T12 is stopped.

Example:

```
RESET            ( Sends Reset packet on currently selected LCN )
CH6  RESET       ( Sends Reset packet on channel 6 )
```

RESTART (--)

If the layer 3 state machine is in a state that allows a restart request packet to be sent, the emulation transmits a restart request packet for a DTE emulation or a restart indication packet for a DCE emulation. This packet is sent out on logical channel zero.

The T20/T10 timer is started and the layer 3 state machine is set to the waiting restart confirm state. In response, the emulation expects a restart confirm packet. If a correctly formatted restart confirm packet is received, the layer 3 state machine goes to the Link Up state and the LCN state machine goes to state p1 – Ready for an SVC or state d1 – Flow Control Ready for a PVC. T20/T10 is stopped and CH1 is selected as the current LCN.

NOTE
RESTART, CALL, DATA, RESET, CLEAR, and INTERRUPT packets can also be transmitted by pressing the corresponding function key under the L3Send topic.

The following two commands transmit user-defined data (80 characters maximum).

SENDP (string --)

Transmits a user-defined packet. The defined string is transmitted in an information field with the correct N(S) and N(R).

SENDD (string --)

Transmits a user-defined data field in a data packet. The defined string is transmitted in a data packet containing the correct P(S) and P(R).

SENDD only transmits a data packet if the corresponding layer 3 window is open and sufficient acknowledgements have been received. The status of the layer 3 window is determined using the WINDOW? command.

If the layer 3 window is not open, the data is discarded as there is no queuing implemented at this level.

Example:
```
X" 10010B22303000000000" SENDP ( Transmits a call request packet on LCN 1 )
" A quick brown fox jumped over the lazy dog" SENDD
            ( Transmits a data packet on current LCN with this text in data field )
```

12.4 State–Independent Send Commands

The following commands transmit frames/packets without regard to correct protocol procedure. They do not update the protocol states or increment sequence values, eg. N(S) and N(R) and are used to generate frames/packets out of context. The variables used in these commands are described in Section 11.

Layer 2

S:SABM (--)

Transmits a SABM command frame using values from the following variables:

YADDR	– Frame address (1 or 3 for SLP, and 7 or 15 for MLP)
PF	– Value of P bit (0 or 1)

The frame layer emulation goes to modulo 8.

Example:
Transmit a SABM frame with an incorrect address byte and P equal to 1.

```
1 PF !              ( Set P bit )
4 YADDR !           ( Set address to illegal value )
S:SABM              ( Send frame )
```

S:SABME (--)

Transmits a SABME command frame using values from the following variables.

YADDR	– Frame address (1 or 3 for SLP, and 7 or 15 for MLP)
PF	– Value of P bit (0 or 1)

The frame layer emulation goes to modulo 128.

S:DISC (--)

Transmits a DISC command frame using values from the following variables:

YADDR	– Frame address (1 or 3 for SLP, and 7 or 15 for MLP)
PF	– Value of P bit (0 or 1)

S:UA (--)

Transmits a UA response frame using values from the following variables:

MADDR	– Frame address (1 or 3 for SLP, and 7 or 15 for MLP)
P/F	– Value of F bit (0 or 1)

S:DM (--)

Transmits a DM response frame using values from the following variables:

MADDR	– Frame address (1 or 3 for SLP, and 7 or 15 for MLP)
P/F	– Value of F bit (0 or 1)

S:FRMR (--)

Transmits an FRMR response frame using values from the following variables:

MADDR	– Frame address (1 or 3 for SLP, and 7 or 15 for MLP)
P/F	– Value of F bit (0 or 1)
CONTROL	– Value of received control byte
VS	– Current send sequence variable value (0 through 7 for modulo 8, and 0 through 127 for modulo 128)
VR	– Current receive sequence variable value (0 through 7 for modulo 8, and 0 through 127 for modulo 128)
C/R	– Last received frame response indicator (0 = command, 1 = response)
FRMRW	– W bit (0 or 1)
FRMRX	– X bit (0 or 1)
FRMRY	– Y bit (0 or 1)
FRMRZ	– Z bit (0 or 1)

S:RR (--)

Transmits an RR response frame using values from the following variables:

MADDR	- Frame address (1 or 3 for SLP, and 7 or 15 for MLP)
P/F	- Value of F bit (0 or 1)
VR	- Value of N(R) sequence number (0 through 7 for modulo 8, and 0 through 127 for modulo 128)

S:RRC (--)

Transmits an RR command frame using values from the following variables:

YADDR	- Frame address (1 or 3 for SLP, and 7 or 15 for MLP)
PF	- Value of P bit (0 or 1)
VR	- Value of N(R) sequence number (0 through 7 for modulo 8, and 0 through 127 for modulo 128)

S:RNR (--)

Transmits an RNR response frame using values from the following variables:

MADDR	- Frame address (1 or 3 for SLP, and 7 or 15 for MLP)
P/F	- Value of F bit (0 or 1)
VR	- Value of N(R) sequence number (0 through 7 for modulo 8, and 0 through 127 for modulo 128)

S:RNRC (--)

Transmits an RNR command frame using values from the following variables:

YADDR	- Frame address (1 or 3 for SLP, and 7 or 15 for MLP)
PF	- Value of P bit (0 or 1)
VR	- Value of N(R) sequence number (0 through 7 for modulo 8, and 0 through 127 for modulo 128)

S:REJ (--)

Transmits an REJ response frame using values from the following variables:

MADDR	- Frame address (1 or 3 for SLP, and 7 or 15 for MLP)
P/F	- Value of F bit (0 or 1)
VR	- Value of N(R) sequence number (0 through 7 for modulo 8, and 0 through 127 for modulo 128)

S:REJC (--)

Transmits an REJ command frame using values from the following variables:

YADDR	- Frame address (1 or 3 for SLP, and 7 or 15 for MLP)
PF	- Value of P bit (0 or 1)
VR	- Value of N(R) sequence number (0 through 7 for modulo 8, and 0 through 127 for modulo 128)

S:INF (--)

Transmits an information command frame using values from the following variables:

YADDR	– Frame address (1 or 3 for SLP, and 7 or 15 for MLP)
PF	– Value of P bit (0 or 1)
VR	– Value of N(R) sequence number (0 through 7 for modulo 8, and 0 through 127 for modulo 128)
VS	– Value of N(S) sequence number (0 through 7)

The information field sent in this frame is the next packet waiting to be transmitted. If no packets are waiting for transmission, the I frame is not transmitted. S:INF increments the VS variable.

NOTE
S:RRC, S:RNRC, S:REJ, S:RR, S:RNR, and S:REJ frames can be transmitted by pressing the corresponding function keys under the L2Send topic.

Layer 3

The send packet commands can be used alone if the layer 2 link protocol state machine is in the information transfer states (states 5 to 8). If not, the send packet commands must be followed by S:INF to send the information frame.

S:RESTARTR (--)

Transmits a restart request packet on logical channel 0 using values from the following variables:

LCN	– Logical group and channel value (0)
SENDQB	– Q bit value
SENDDB	– D bit value
SENDGFI	– GFI modulo value (0x10 for modulo 8, and 0x20 for modulo 128)
SCAUSE	– Cause value
SDIAG	– Diagnostic value

S:RESTARTC (--)

Transmits a restart confirm packet on logical channel 0 using values from the following variables:

LCN	– Logical group and channel value (0)
SENDQB	– Q bit value
SENDDB	– D bit value
SENDGFI	– GFI modulo value (0x10 for modulo 8, and 0x20 for modulo 128)

S:CALLR (--)

Transmits a call request/incoming call packet on the selected channel using values from the following variables:

LCN	- Logical group and channel value
SENDQB	- Q bit value
SENDDB	- D bit value
SENDGFI	- GFI modulo value (0x10 for modulo 8, and 0x20 for modulo 128)
LCNCALLED	- Called address field with the first byte containing the length of address and followed by the address bytes in ASCII
LCNCALLING	- Calling address field with the first byte containing the length of address and followed by the address bytes in ASCII
FACILITY-ACTIVE	- Indication of type of facility negotiation desired 0 = Facilities not wanted 1 = Negotiate facilities for size, window, and throughput class 2 = Send facilities defined with MAKE_FAC command.
CUD-BUFFER	- Call user data field buffer. The first byte is the length of the call user data field followed by the call user data

S:CALLC (--)

Transmits a call connected/accepted packet on selected channel using values from the following variables:

LCN	- Logical group and channel value
SENDQB	- Q bit value
SENDDB	- D bit value
SENDGFI	- GFI modulo value (0x10 for modulo 8, and 0x20 for modulo 128)
RCALLED	- Called address field with the first byte containing the length of address and followed by the address bytes in ASCII
RCALLING	- Calling address field with the first byte containing the length of address and followed by the address bytes in ASCII
CAFAC-ACTIVE	- Indication of the type of facility to use: 0 = Do not send facilities 1 = Echo received facilities from buffer RFAC 2 = Send facilities defined by MAKE_CAFAC command

S:CLEARR (--)

Transmits a clear request/indication packet on the selected channel using values from the following variables:

LCN	- Logical group and channel value
SENDQB	- Q bit value
SENDDB	- D bit value
SENDGFI	- GFI modulo value (0x10 for modulo 8, and 0x20 for modulo 128)
SCAUSE	- Cause value
SDIAG	- Diagnostic value

When the fast select facility is active, the following are also included in clear request packets.

LCNCALLED	- Called address field
LCNCALLING	- Calling address field
FACILITY-ACTIVE	- Indication of type of facility negotiation desired (see S:CALLR)
CUD-BUFFER	- Call user data field buffer. The first byte is the length of the call user data field followed by the call user data

S:CLEARC (--)

Transmits a clear confirmation packet on the selected channel using values from the following variables:

LCN	– Logical group and channel value
SENDQB	– Q bit value
SENDDB	– D bit value
SENDGFI	– GFI modulo value (0x10 for modulo 8, and 0x20 for modulo 128)

S:RESETR (--)

Transmits a reset request/indication packet on the selected channel using values from the following variables:

LCN	– Logical group and channel value
SENDQB	– Q bit value
SENDDB	– D bit value
SENDGFI	– GFI modulo value (0x10 for modulo 8, and 0x20 for modulo 128)
SCAUSE	– Cause value
SDIAG	– Diagnostic value

S:RESETC (--)

Transmits a reset confirmation packet on the selected channel using values from the following variables:

LCN	– Logical group and channel value
SENDQB	– Q bit value
SENDDB	– D bit value
SENDGFI	– GFI modulo value (0x10 for modulo 8, and 0x20 for modulo 128)

S:INTR (--)

Transmits an interrupt packet on the selected channel using values from the following variables:

LCN	– Logical group and channel value
SENDQB	– Q bit value
SENDDB	– D bit value
SENDGFI	– GFI modulo value (0x10 for modulo 8, and 0x20 for modulo 128)
SIUD	– Interrupt user data byte

S:INTC (--)

Transmits an interrupt confirmation packet on the selected channel using values from the following variables:

LCN	– Logical group and channel value
SENDQB	– Q bit value
SENDDB	– D bit value
SENDGFI	– GFI modulo value (0x10 for modulo 8, and 0x20 for modulo 128)

S:DATAP (--)

Transmits a data packet on the selected channel using values from the following variables:

LCN	– Logical group and channel value
SENDQB	– Q bit value
SENDDB	– D bit value
SENDGFI	– GFI modulo value (0x10 for modulo 8, and 0x20 for modulo 128)
NVS	– P(S) value
NVR	– P(R) value
SENDM	– More bit value
DATA-BUF-ID	– Indication of which data buffer to use in user data field 10 = buffer created with SENDD command (maximum length is 80 characters) 11 = echo of received data packets Any other value – DATA-BUFFER which contains the default data field or data field created with DEFINE_DATA command (maximum 4100 characters)
LCNSIZE	– Length of the data field to send when DATA-BUFFER is used
MAX_LENGTH	– Maximum frame length

S:RRP (--)

Transmits an RR packet on the selected channel using values from the following variables:

LCN	– Logical group and channel value
SENDQB	– Q bit value
SENDDB	– D bit value
SENDGFI	– GFI modulo value (0x10 for modulo 8, and 0x20 for modulo 128)
NVR	– P(R) value

S:RNRP (--)

Transmits an RNR packet on the selected channel using values from the following variables:

LCN	– Logical group and channel value
SENDQB	– Q bit value
SENDDB	– D bit value
SENDGFI	– GFI modulo value (0x10 for modulo 8, and 0x20 for modulo 128)
NVR	– P(R) value

S:REJP (--)

Transmits an REJ packet on the selected channel using values from the following variables:

LCN	– Logical group and channel value
SENDQB	– Q bit value
SENDDB	– D bit value
SENDGFI	– GFI modulo value (0x10 for modulo 8, and 0x20 for modulo 128)
NVR	– P(R) value

S:DIAG (--)

Transmits a Diagnostic packet on logical channel 0 using values from the following variables:

LCN	– Logical group and channel value
SENDQB	– Q bit value
SENDDB	– D bit value
SENDGFI	– GFI modulo value (0x10 for modulo 8, and 0x20 for modulo 128)
SDIAG	– Diagnostic value
SDIAG-EXP	– Diagnostic explanation field. The first byte is the length of the field followed by up to three bytes for the explanation field to be sent.

NOTE
S:RRP, S:RNRP, S: REJP, S:DIAG, S:RESTARTC, and S:CALLC can also be transmitted by pressing the corresponding function key under the L3Send topic.

S:REGISTR (--)

Transmits a registration packet on logical channel 0 using values from the following variables:

LCN	– Logical group and channel value
SENDQB	– Q bit value
SENDDB	– D bit value
SENDGFI	– GFI modulo value (0x10 for modulo 8, and 0x20 for modulo 128)
LCNCALLED	– Called address field
LCNCALLING	– Calling address field

NOTE
Registration defined by MAKE_RFAC.

S:REGISTC (--)

Transmits a registration confirmation on logical channel 0 using values from the following variables:

LCN	– Logical group and channel value
SENDQB	– Q bit value
SENDDB	– D bit value
SENDGFI	– GFI modulo value (0x10 for modulo 8, and 0x20 for modulo 128)
RCALLED	– Called address field with the first byte containing the length of address and followed by the address bytes in ASCII
RCALLING	– Calling address field with the first byte containing the length of address and followed by the address bytes in ASCII
SCAUSE	– Cause value
SDIAG	– Diagnostic value

NOTE
Registration defined by MAKE_RFAC.

12.5 CRC Errors

Frames and packets can be sent with correct or incorrect CRC's (FCS), or can be aborted during transmission.

CRC_ERROR (--)
Transmits the next frame or packet with a CRC error. Subsequent frames and packets will be sent correctly.

DO_ABORT (--)
Aborts the next transmitted frame or packet. Subsequent frames and packets will be sent correctly.

GOOD_CRC (--)
Transmits the next frame or packet correctly.

12.6 Multilink Procedure Send and Receive Commands

With the X.25 Emulation, the user can build, send, receive, and decode X.25 frames with the MLP layer present. No automatic MLP emulation exists. MLP can be simulated using test scripts which:

- define the multilink control field using MLP send commands; and
- transmit frames including the multilink control field by using any of the single link send packet commands such as SENDP, SENDD, CALL, CLEAR, etc., or the SENDMLP command.

NOTE
The contents of the multilink control field are not changed automatically by the program, but *must be defined in the test script prior to transmitting a frame.*

Multilink Control Field Format

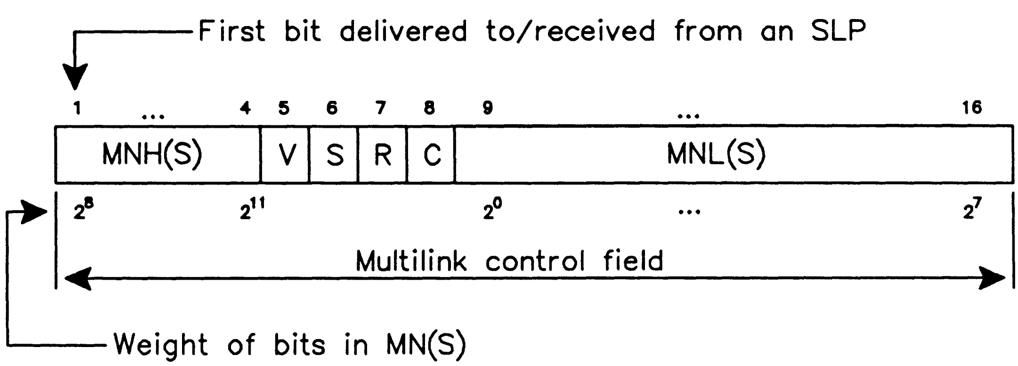


Figure 12–1 Multilink Control Field Format

MNH(S) and MNL(S) The 12 bit multilink sequence number MN(S) is split into two fields. MNH(S) contains bits 9 to 12, and MNL(S) contains bits 1 to 8. Their combined value gives valid sequence numbers for multilink frames of 0 to 4095. In a true emulation, this value increments by 1 each time a multilink frame is received.

V The void sequencing bit contains a 0 when sequencing is required and 1 when sequencing is not required.

S The sequence check option bit is significant only when the V bit is set to 1 indicating that sequencing is not required. It contains 0 when an MN(S) number has been assigned for multilink frame checking and a 1 when no MN(S) number has been assigned.

R The reset request bit contains 0 for normal communications. When the R bit contains a 1, the emulation is requesting a multilink reset. No packet level information is carried in this frame but an optional one bit cause field can be present.

C The reset confirmation bit contains 0 for normal communications. This bit contains a 1 when the frame is a response to an MLP frame received with the R bit set to 1, indicating that the reset process has been confirmed. No information field is included when the C bit contains a 1.

Multilink Receive Commands

GET_MNS (-- MNS value)
Obtains the MN(S) field from the last received MLP frame. Valid values are 0 through 4095.

GET_V (-- V bit)
Obtains the value contained in the V bit of the last received MLP frame. Valid values are 0 indicating that sequencing is required, and 1 indicating that sequencing is not required.

GET_S (-- S bit)
Obtains the value contained in the S bit of the last received MLP frame. Valid values are 0, indicating that MN(S) sequence number has been assigned, and 1 indicating that no MN(S) has been assigned.

GET_R (-- R bit)
Obtains the value contained in the R bit of the last received MLP frame. Valid values are 0 indicating normal communication, and 1 requesting a multilink reset.

GET_C (-- C bit)
Obtains the value contained in the C bit of the last received MLP frame. Valid values are 0 indicating normal communication, and 1 responding to a request for a multilink reset.

CAUSE_RECEIVED? (-- flag)
Returns true if the last received MLP frame contained a reset cause field. When the flag equals 1, the cause field can be obtained using the GET_CAUSE command.

GET_CAUSE (-- reset cause field)

Obtains the reset cause field from the last received MLP frame. Valid values are 0 through 255. Use only when CAUSE_RECEIVED? returns a 1 on the stack.

Example:
Obtain the reset cause field from the last received frame.

```
CAUSE_RECEIVED?                          ( Field present ? )
IF                                       ( Yes )
      GET_CAUSE                          ( Obtain the cause )
      T." MLP reset cause field =" T. TCR ( Create a trace statement )
ENDIF
```

Multilink Send Commands

LOAD_MNS (MNS value --)

Sets the MN(S) value (divided properly into the MNH(S) and MNL(S) fields) for subsequent transmitted MLP frames. This value for MN(S) remains in effect until LOAD_MNS is called again, i.e. no automatic sequence numbering occurs in the emulation. Valid values are 0 (default) through 4095. This MN(S) value is stored in the SMNS variable.

SMNS (--address)

Contains the MN(S) used when MLP frames are transmitted. Valid values are 0 (default) through 4095.

Example:
Obtain the last transmitted MN(S) value and increment this value for the next MLP frame to be transmitted. This value must not exceed 4095 (12 bits).

```
SMNS @          ( Get previous value in SMNS variable )
1+              ( Increment )
4095 AND        ( Use modulo 4095 - 12 bits )
LOAD_MNS        ( Store incremented value back in SMNS )
```

LOAD_V (V bit value --)

Determines the value of the V bit for subsequent transmitted MLP frames. This value remains in effect until changed by another LOAD_V command. Valid values are 0 (default) and 1.

LOAD_S (S bit value --)

Determines the value of the S bit for subsequent transmitted MLP frames. This value remains in effect until changed by another LOAD_S command. Valid values are 0 (default) and 1.

LOAD_R (R bit value --)

Determines the value of the R bit for subsequent transmitted MLP frames. This value remains in effect until changed by another LOAD_R command. Valid values are 0 (default) and 1.

LOAD_C (C bit value --)

Determines the value of the C bit for subsequent transmitted MLP frames. This value remains in effect until changed by another LOAD_C command. Valid values are 0 (default) and 1.

LOAD_CAUSE (cause field --)
Sets the contents of the reset cause field used in a multilink frame when the R bit is set to one. This value remains in effect until changed by another LOAD_CAUSE command. Valid values are 0 (default) through 255.

SEND_CAUSE (1|0 --)
Determines if the reset cause field is included when multilink frames are sent with the R bit set to 1. When the input parameter is set to 1, all subsequent multilink frames transmitted via the SENDMLP command include the reset cause field as last set by the LOAD_CAUSE command.

SENDMLP (--)
Transmits a multilink I frame with the MLP control field constructed according to the LOAD_x commands. No packet layer is included. SENDMLP is useful for multilink resetting or confirmation of resetting.

Example:
Transmit a multilink reset request including the reset cause field.

```
SNMS @              ( Get previous value in SMNS variable )
1+                  ( Increment )
4095 AND            ( Use modulo 4095 - 12 bits )
LOAD_MNS            ( Store incremented value back in SMNS )
1 LOAD_R            ( Set the reset request bit )
8 LOAD_CAUSE        ( Define a cause field )
1 SEND_CAUSE        ( Include the reset cause field )
SENDMLP             ( Transmit the frame )
```

13
TEST MANAGER

IDACOM has developed a comprehensive set of tools for the development of test scripts. These test scripts, written using the ITL language, control the operation of the X.25 Monitor/Emulation application.

For a complete explanation of the test manager and tools available, see the Programmer's Reference Manual.

This section reviews basic ITL components and describes the protocol event and action commands specific to X.25.

13.1 ITL Constructs

Following is a brief description of test manager constructs. For more details and examples, refer to the Programmer's Reference Manual.

TCLR (--)
Initializes the test manager. Any existing test suites already in memory are cleared. The current state is set to 0. All test scenarios should start with the TCLR command.

STATE_INIT{ }STATE_INIT (number --)
Brackets the execution sequence performed prior to entering a state. The initialization logic for a state is executed independently of how it was called.

This initialization procedure can be used for any state but is not compulsory. STATE_INIT{ must be preceded by the number of the state being initialized, eg. 0 STATE_INIT{.

The STATE_INIT{ }STATE_INIT clause is executed only once each time the state is entered from another state.

STATE{ }STATE (number --)
Brackets a state definition. STATE{ must be preceded by the number of the state. Valid values are 0 through 255. State 0 must be defined within an ITL program. If not, the test manager will not run the script. If multiple states are defined with the same number in the test script, the test manager uses the latest definition.

ACTION{ }ACTION (f --)
Brackets the set of tasks, decisions, and outputs which execute once the expected event is received by the test manager. There must be at least one action defined for each expected event. The action is executed when the flag is true (non zero).

NEW_STATE (n --)
Executes the initialization logic of the specified state (providing STAT_INIT{ }STAT_INIT is defined) and establishes the state to be executed for the next event. Any remaining action code for the current state is then executed. It must be preceded with a valid state number and be inside the ACTION{ }ACTION brackets. This command is not mandatory if no state change is desired.

TM_STOP (--)
Stops the execution of the test script. The test suite remains in memory and can be re-executed until another test script is loaded.

SEQ{ }SEQ (number --)
Brackets a definition of tasks and outputs which execute as part of the state machine action. SEQ{ expects a single integer which is the sequence number. Up to 256 sequences are supported. Valid values are 0 through 255. The SEQ{ }SEQ partners are extremely useful when more than one action sequence calls the same tasks and outputs. The SEQ{ }SEQ definition is defined outside the ACTION{ }ACTION definition and then called by the RUN_SEQ command.

This is an alternate mechanism to generate colon definitions. This mechanism causes the equivalent of a colon definition (now accessed via a numeric identifier) to be compiled into the test script dictionary rather than the user dictionary. Refer to the Programmer's Reference Manual.

RUN_SEQ (number --)
Executes a specified set of tasks defined in a SEQ{ }SEQ definition. It is called inside an ACTION{ }ACTION definition and must be preceded with a defined sequence number.

LOAD_RETURN_STATE (number --)
Permits the test script writer to program the equivalent of subroutine calls (used with RETURN_STATE). LOAD_RETURN_STATE sets the state to which control is to be returned. LOAD_RETURN_STATE must be within the action field; nesting is not permitted.

RETURN_STATE (--)
Returns control to the state specified by LOAD_RETURN_STATE from a state subroutine call.

NEW_TM (filename --)
Loads and compiles the specified file and then starts the test manager at state 0. It can be included as part of the action field to load and execute another scenario.

13.2 Event Recognition

During test script execution, any event received by the test manager is evaluated to determine if it matches the event-specifier of the first action within that state. If the evaluation does not return true, the following action clauses are evaluated in a sequential manner. Once an event evaluates true, the subsequent action clauses in that particular state are not examined.

NOTE
In automatic emulation, if the test manager is running and the ?RX event recognition is not in use, the following line is required within the state to process the received frame or packet.

```
NO 1 ?RX ACTION{  }ACTION
```

Event Recognition Commands	Layer 2	Layer 3
?RX	YES	YES
?RX_FRAME	NO	NO
?RX_PACKET	YES	NO

Table 13–1 Test Manager and Automatic Emulation Interaction

NOTE
See the individual commands in the next section for detailed descriptions.

Layer 1

If the X.25 Monitor or Emulation is running on a B-Channel or PRA Test Channel, no layer 1 events will be received by the test manager. See the Programmer's Reference Manual for a description of layer 1 events, i.e. control lead transitions when the application is running on a WAN interface.

NOTE
Interface leads must be enabled.

Received Frames

ITL provides recognition of protocol specific frames, packets, anchored or unanchored comparison of user-specified octets, CRC errors, and aborted frames.

Any frames received by the monitor or emulation are decoded and the decoded information is stored in various communication variables. The decoded information is used by the test manager to identify a particular event and can be obtained by the test program with the @ (fetch) command.

An identifier stored in the FRAME-TYPE variable is associated with each frame.

Frame Identifiers	Description
R*SABM	Set asynchronous balanced command frame
R*SABME	Set asynchronous balanced extended command frame
R*DISC	Disconnect command frame
R*I	Information frame
R*RRC	Receive ready command frame
R*RNRC	Receive not ready command frame
R*REJC	Reject command frame
R*RR	Receive ready response frame
R*RNR	Receive not ready response frame
R*REJ	Reject response frame
R*UA	Unnumbered acknowledgment response frame
R*DM	Disconnected mode response frame
R*FRMR	Frame reject response frame
R*INVFRM	Invalid frame

Table 13–2 Frame Identifiers

If the frame is an information frame, the packet type is stored in the PACKET-TYPE variable. The identifiers used to match received packets are listed in Table 13-3.

Packet Identifiers	Description
R*DATAP	Data packet
R*RRP	Receive ready packet
R*RNRP	Receive not ready packet
R*REJP	Reject packet
R*CALLREQ	Call request or incoming call packet
R*CALLCON	Call connect or call accept packet
R*CLEARREQ	Clear request or clear indication packet
R*CLEARCONF	Clear confirm packet
R*INTREQ	Interrupt packet
R*INTCONF	Interrupt confirm packet
R*RESETREQ	Reset request or reset indication packet
R*RESETCONF	Reset confirm packet
R*RESTARTREQ	Restart request or restart indication packet
R*RESTARTCONF	Restart confirm packet
R*DIAGNOSTIC	Diagnostic packet
R*REGISTREQ	Registration request packet
R*REGISTCONF	Registration confirm packet
R*INVPKT	Invalid packet

Table 13-3 Packet Identifiers

NOTE
If the received frame does not contain a packet, the PACKET-TYPE variable contains 0.

For a more complete list of decoded communication variables, refer to Sections 4 and 11.

?FRAME (-- flag)
Returns true if any frame is received.

Example:
In state 0, the reception of any frame results in a message to the user and the script proceeds to state 1.

```
0 STATE{
     ?FRAME                        ( Don't care what type of frame )
     ACTION{
         T." First frame received.  Test is starting." TCR
         1  NEW_STATE
     }ACTION
  }STATE
```

?PACKET (-- flag)
Returns true if an I frame is received.

?RX_FRAME (frame id #1\...\frame id#n\n -- flag)
Returns true if one of the specified frames is received. See Table 13-2 for a list of valid frame identifiers.

NOTE
In the emulation, to allow the layer 2 state machine to respond automatically when using ?RX_FRAME, the RUN_LAYER2 command must be issued within the action sequence. See Table 13-1.

Example:
Look for the reception of any of the supervisory frames using the monitor. The user receives an audible alarm and the test manager goes to state 2.

```
1 STATE{
      R*RRC R*RR R*RNRC R*RNR R*REJC R*REJ  6  ?RX_FRAME
      ACTION{
          BEEP
          2 NEW_STATE
      }ACTION
  }STATE
```

Example:
Look for reception of either a UA or DM frame using the emulation. The automatic layer 2 state machine is called with the RUN_LAYER2 command and a test manager state change occurs.

```
10 STATE{
      R*UA R*DM  2  ?RX_FRAME
      ACTION{
          RUN_LAYER2                     ( Run automatic layer 2 )
          11  NEW_STATE                  ( Go to state 11 )
      }ACTION
   }STATE
```

Example:
Look for the reception of a DM frame using the emulation. The automatic layer 2 state machine is not used. A SABM frame is sent and a test manager state change occurs.

```
5 STATE{
      R*DM  1  ?RX_FRAME
      ACTION{
          S:SABM                         ( Send SABM )
          6  NEW_STATE                   ( Go to state 6 )
      }ACTION
  }STATE
```

?RX_PACKET (packet id #1\...\packet id #n\n -- flag)

Returns true if one of the specified packets is received. See Table 13-3 for a list of valid packet identifiers.

NOTE

In the emulation, the layer 2 state machine is run automatically when this command is used. The layer 3 state machine is not executed. All layer 3 responses must be generated within the action sequences. See Table 13-1.

Example:
Look for the reception of eight packets which can be either a data or RR packet using the monitor. After eight matching packets are received, the test manager goes to state 2.

```
0 STATE{
      ?WAKEUP                           ( Wakeup timer received? )
      ACTION{
          0 COUNTER !                   ( Reset counter )
          1 NEW_STATE
      }ACTION
  }STATE

1 STATE{
      R*DATAP R*RRP  2  ?RX_PACKET
      ACTION{
          1 COUNTER  +!                 ( Increment counter )
          COUNTER @ 8 =                 ( Have we received 8? )
          IF                            ( Yes )
              2 NEW_STATE               ( Go to state 2 )
          ENDIF
      }ACTION
  }STATE
```

Example:
Look for the reception of an RR packet. When one is received, a data packet is sent.

```
2 STATE{
      R*RRP  1  ?RX_PACKET
      ACTION{
          DATA                          ( Send a data packet )
      }ACTION
  }STATE
```

?RX (frame id#1 or packet id #1\...\frame id#n or packet id#n\n -- flag)
Returns true if one of the specified frames or packets is received. See Tables 13-2 and 13-3 for valid frame and packet identifiers.

NOTE
In the emulation, both layer 2 and layer 3 state machines are executed automatically. See Table 13-1.

Example:
Look for reception of invalid frames or packets using the monitor. The user receives an audible alarm and a notice.

```
3 STATE{
      R*INVFRM R*INVPKT  2  ?RX
      ACTION{
          BEEP
          " Invalid frame or packet received." W.NOTICE
      }ACTION
  }STATE
```

Example:
Look for the reception of a call request packet using the emulation. The automatic emulation provides the response and the test manager proceeds to state 6.

```
5 STATE{
      R*CALLREQ  1  ?RX
      ACTION{
          6  NEW_STATE              ( Layer 3 state machine provides response )
      }ACTION
  }STATE
```

?RX_DATA (string -- flag)
Returns true if a user-defined character string is found in the data field of received packets.

This is an *anchored* match, i.e. a byte-for-byte match starting at the first byte of the data field of a received data packet.

NOTE
To accommodate "don't care" character positions, the question mark character for ASCII or hex 3F character can be used. The specified string is limited to 80 characters. The received data field can be longer than the specified string.

Example:
Search for the string 'IDACOM' starting at the second byte of a data packet. The first byte is a "don't care" and matches on any value.

```
" ?IDACOM"  ?RX_DATA
```

or

```
X" 3F494441434F4D"  ?RX_DATA
```

?RECEIVED (string -- flag)
Returns true if a user-defined character string is found in the received frame.

This is an *anchored* match, i.e. a byte-for-byte match starting at the first byte of the received frame.

Example:
Detect a DISC frame with the P bit set.

```
X" 3F53"  ?RECEIVED
```

The control byte for a DISC frame with the P bit set would contain the hex value of 53.

NOTE
To accommodate "don't care" character positions, the question mark character for ASCII or hex 3F character can be used.

?SEARCH (string -- flag)
Returns true if a user-defined character string is found in the received frame.

This is an *unanchored* match, i.e. searches for an exact match anywhere in the received frame, regardless of position.

Example:
Search for the string 'IDACOM' which could be located starting at any position within the received frame.

```
" IDACOM"  ?SEARCH
```

?ABORT (-- flag)
Returns true if an abort frame is received.

?CRC_ERROR (-- flag)
Returns true if a frame with a CRC error is received.

WINDOW? (-- flag)
Calculates the window for the selected logical channel. Returns 0 if the window is closed and 1 if the window is open.

Example:
Send a data packet while the window open.

```
BEGIN
      WINDOW?                        ( Check LCN Data Window )
WHILE                                ( While window is open )
      DATA                           ( Send a data packet )
REPEAT                               ( Repeat until window is closed )
```

Timeout Detection

There are 128 user programmable timers available. Timers 1 through 24 can be used in the test manager. Timer 34 is the test manager wakeup timer. The remaining timers are used in the application and should not be started or stopped in a test script.

TIMEOUT (--flag)
Returns true if any timer has expired.

Example:
In State 8, look for the expiration of any timer. The action is to display a trace statement.

```
8 STATE{
    TIMEOUT        ( Check for timeout of any timer )
    ACTION{
        T." A Timer has expired." TCR
    }ACTION
}STATE
```

?TIMER (timer # -- flag)
Returns true if the specified timer has expired. Valid input parameters are timers 1 through 24.

Example:
In State 8, look for the expiration of timer 21. The action is to display a trace statement.

```
8 STATE{
    21 ?TIMER          ( Check for timeout of timer 21 )
    ACTION{
        T." Timer 21 has expired." TCR
    }ACTION
}STATE
```

?WAKEUP (-- flag)

Returns true if the wakeup timer has expired. The wakeup timer can be used to initiate action sequences immediately upon the test manager starting. Timer 34 is started for 100 milliseconds when the test manager is started after a WAKEUP_ON command has been issued. The default is WAKEUP_OFF.

Example:
In State 0, look for the expiration of the wakeup timer. The action is to prompt the user to press a function key, and then the test manager goes to State 1.

```
0 STATE{
      ?WAKEUP        ( Check for timeout of wakeup timer )
      ACTION{
          T." To start the test, press UF1." TCR
          1 NEW_STATE
      }ACTION
  }STATE
```

TIMER–NUMBER (-- address)

Contains the number of the expired timer. Valid values are 1 through 128.

T1–TIMER (-- value)

A constant identifier for the T1 timer which can be used with either the TIMEOUT or ?TIMER commands.

Example:
Detect the T1 timer timeout using TIMEOUT.

```
5 STATE{
      TIMEOUT
      TIMER-NUMBER @  T1-TIMER =  AND            ( T1 Timeout detected )
      ACTION{
          T." T1 Timer has timed out."  TCR      ( Advise the user )
      }ACTION
  }STATE
```

Example:
Detect the T1 timer timeout using the ?TIMER command.

```
5 STATE{
      T1-TIMER  ?TIMER                           ( T1 timer detected )
      ACTION{
          T." T1 Timer has timed out."  TCR      ( Advise the user )
      }ACTION
  }STATE
```

NOTE
The two previous examples provide identical actions. The ?TIMER example is a more efficient method of coding.

Example:
Detect the channel 0 timer used with restart packets.

```
6 STATE{
      30  ?TIMER                    ( Channel 0, restart timer? )
      ACTION{
          T." Restart timer has timed out." TCR
      }ACTION
  }STATE
```

LCNTIMER (-- address)
Contains the timer value for the selected LCN.

Example:
Detect a timeout on channel 50.

```
TIMEOUT
TIMER-NUMBER @ CH50 LCNTIMER @ =
AND
ACTION{
      ...
}ACTION
```

or

```
CH50  LCNTIMER  @  ?TIMER
ACTION{
      ...
}ACTION
```

NOTE
The two previous examples provide identical actions. The ?TIMER example is a more efficient method of coding.

Function Key Detection

Refer to the Programmer's Reference Manual.

Interprocessor Mail Events

Refer to the Programmer's Reference Manual.

Wildcard Events

The X.25 application supports the OTHER_EVENT command and the EVENT–TYPE variable. Refer to the Programmer's Reference Manual.

The EVENT–TYPE variable contains any one of the following constants: FRAME, LEAD*CHANGE, TIME*OUT, FUNCTION*KEY, or COMMAND_IND.

FRAME (-- value)
A constant value in the EVENT–TYPE variable when the received event is a frame. The actual protocol type is in either the FRAME–TYPE and PACKET–TYPE variables. See the 'Received Frames' section on page 13–4.

LEAD*CHANGE (-- value)
A constant value in the EVENT–TYPE variable when the received event is a control lead transition. The actual lead transition is in the LEAD–NUMBER variable.

TIME*OUT (-- value)
A constant value in the EVENT–TYPE variable when the received frame is a timeout. The actual timer is in the TIMER–NUMBER variable. See the 'Timeout Detection' section on page 13–10.

FUNCTION*KEY (-- value)
A constant value in the EVENT–TYPE variable when a function or cursor key is detected. The actual key value is in the KEY–NUMBER variable.

NOTE
To detect function keys, it is advisable to use the ?KEY command. Refer to the Programmer's Reference Manual.

COMMAND_IND (-- value)
A constant value in the EVENT–TYPE variable when an interprocessor mail indication is received. Refer to the Programmer's Reference Manual.

13.3 X.25 Actions

All of the general actions explained in the Programmer's Reference Manual are supported in the X.25 Monitor and Emulation. For additional display commands specific to X.25, refer to Section 7 of this manual.

Layer 1 Actions

The following emulation commands turn control leads on and off.

V.28/RS–232C Interface		
OFF to ON	ON to OFF	Description
RTS_ON	RTS_OFF	Request to send
CTS_ON	CTS_OFF	Clear to send
DSR_ON	DSR_OFF	Data set ready
CD_ON	CD_OFF	Carrier detect
DTR_ON	DTR_OFF	Data terminal ready
SQ_ON	SQ_OFF	Signal quality
RI_ON	RI_OFF	Ring indicate
DRS_ON	DRS_OFF	Data signal rate select

Table 13–4 V.28/RS–232C Interface Lead Transitions

V.35 Interface		
OFF to ON	ON to OFF	Description
RTS_ON	RTS_OFF	Request to send
CTS_ON	CTS_OFF	Clear to send
DSR_ON	DSR_OFF	Data set ready
CD_ON	CD_OFF	Carrier detect
DTR_ON	DTR_OFF	Data terminal ready
RI_ON	RI_OFF	Ring indicate

Table 13–5 V.35 Interface Lead Transitions

V.36/RS–449 Interface		
OFF to ON	ON to OFF	Description
RTS_ON	RTS_OFF	Request to send
CTS_ON	CTS_OFF	Clear to send
DSR_ON	DSR_OFF	Data set ready
DTR_ON	DTR_OFF	Data terminal ready
RI_ON	RI_OFF	Calling indicator
DRS_ON	DRS_OFF	Data signal rate select
CD_ON	CD_OFF	Data channel received line signal

Table 13–6 V.36/RS–449 Interface Lead Transitions

Protocol Actions

Frames or packets can be transmitted using any of the commands described in Section 12. The test manager or the automatic protocol state machine provides the response according to the command chosen for event recognition.

NOTE
The test manager can be run with the layer 2 and layer 3 state machines running automatically or turned off. If the state machines are set to automatic mode, the programmer should choose the test manager commands carefully. See Table 13–1 on page 13–3.

RUN_LAYER2 (--)
Executes the layer 2 state machine to process the received event, makes a state transition (if applicable), and shuts down timers that have been started. RUN_LAYER2 is intended for use when only layer 2 testing is performed. See ?RX_FRAME for an example.

For examples of transmitting frames and packets using the test manager, see Section 14.

Using Buffers

IDACOM's test manager has 256 buffers available for creating customized frames. These buffers are numbered from 0 through 255 and can be created any size desired. However, the X.25 Emulation limits the number of bytes that can be transmitted to 4110.

A buffer consists of four bytes with values of 0, two bytes containing the length of the text, and the remaining bytes consisting of user-defined text.

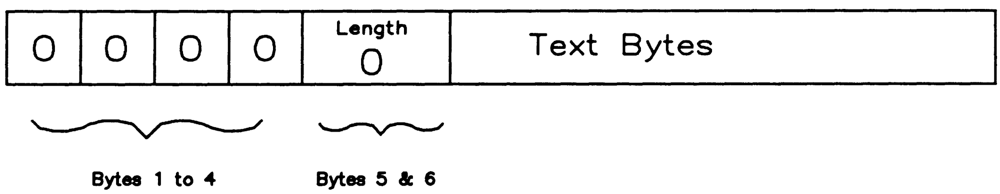


Figure 13–1 Buffer Structure

NOTE
All buffers are cleared when the TCLR command is issued. TCLR is usually the first command compiled when loading a test script.

There are three methods of moving text into a buffer.

Methods 1 and 2 automatically allocate memory for the specified text. Method 3 requires the user to allocate memory before moving text into the buffer. Use the TCLR command to clear all buffers.

Method 1

STRING->BUFFER (string\buffer number --)
Loads a quoted string into the specified buffer. The length is limited to 80 bytes if typing directly on the keyboard and 255 bytes if used within a test script. Either an ASCII or hex string can be specified. Valid buffer numbers are 0 through 255.

Example:

```
" IDACOM" 1 STRING->BUFFER          ( ASCII text moved to Buffer #1 )
X" 0100100100434445" 2 STRING->BUFFER ( Hex string of 8 bytes moved to Buffer #2 )
```

Method 2

FILE->BUFFER (filename\buffer number --)
Transfers a text file into the specified buffer (for text greater than 80 bytes). The file is created using the Edit function available on the Home processor. At this time, only ASCII text can be created. The last character to be transferred should be followed immediately by a CTRL 'p' character in the file. This special character is displayed as a pilcrow (¶) character. The file is transferred into the buffer until the ASCII control 'p' character is found or until the end of the file.

Example:
Create a file with the name CUSTOM.F and transfer to Buffer #3.

```
" CUSTOM.F" 3 FILE->BUFFER
```

Method 3

The following commands should not be used with FILE->BUFFER or STRING->BUFFER.

ALLOT_BUFFER (size \ buffer number -- flag)
Allocates memory for the specified buffer. ALLOT_BUFFER returns 0 if an error occurred, or 1 if correct.

NOTE
ALLOT_BUFFER should not be used repetitively with the same buffer number in the same test script.

FILL_BUFFER (data address \ size \ buffer number --)
Moves data, of a specified size, into a buffer. Previous contents are overwritten.

APPEND_TO_BUFFER (data address \ size \ buffer number --)
Appends data, of a specified size, into a buffer.

CLEAR_BUFFER (buffer number --)
Stores a size of 0 in the buffer. CLEAR_BUFFER has no effect on the allocated memory defined with ALLOT_BUFFER.

Example:
```
0 VARIABLE tempstring 6 ALLOT
" A TEST " tempstring $!                ( Initialize the string )
16 3 ALLOT_BUFFER                       ( Allocate 16 bytes of memory )
IF
      tempstring 4+ 5 3 FILL_BUFFER     ( Move 'TEST ' to buffer )
      " FAIL" COUNT 3 APPEND_TO_BUFFER  ( Append 'FAIL' to buffer )
ENDIF
```

BUFFER (buffer number -- address | 0)
Returns the address of the first byte of the specified buffer. The buffer must have been previously created by FILE->BUFFER, STRING->BUFFER, or ALLOT_BUFFER. A '0' is returned when the buffer is not created or an invalid buffer number is specified. Valid buffer numbers are 0 through 255.

Sending a Buffer

The text must first be stored in the buffer using STRING->BUFFER or FILE->BUFFER. Once the text is in place, the buffer can be transmitted repetitively.

The actual size of the frame or packet sent is defined by the frame and packet size set on the Frame and Packet Layer Menus or by the value stored in the MAX_LENGTH variable and the =SIZE command.

BUFFER_SENDF (buffer number --)
Transmits the specified buffer as an entire frame. Valid buffer numbers are 0 through 255.

Example:
Create the text to be included in the buffer first and then transmit the buffer.
```
X" 0100100100434445" 2 STRING->BUFFER      ( Create text )
2  BUFFER_SENDF                            ( Send buffer )
```

NOTE
When using BUFFER_SENDF, the first byte of the buffer defines the frame address and the second byte the frame control.

BUFFER_SENDP (buffer number --)
Transmits the specified buffer as the packet layer carried by an information frame. Valid buffer numbers are 0 through 255.

Example:
Create the text in the buffer and then transmit the buffer.
```
X" 10010043445"  3  STRING->BUFFER         ( Create text )
3  BUFFER_SENDP                            ( Send packet )
```

BUFFER_SENDD (buffer number --)

Transmits the specified buffer as the data field of a data packet. Valid buffer numbers are 0 through 255.

Example:
Create the text in the buffer and then transmit the buffer.

```
" IDACOM" 4 STRING->BUFFER              ( Create text )
4 BUFFER_SENDD                          ( Send data packet )
```

Example:
The text to be included is longer than 80 characters and is in a file named CUSTOM.F.

```
" CUSTOM.F" 5 FILE->BUFFER              ( Put text in buffer )
5 BUFFER_SENDD                          ( Send Data packet )
```

14
TEST SCRIPTS

This section contains sample complete test scripts. These test scripts have also been supplied on disk and can be loaded and run as described in the Programmer's Reference Manual.

14.1 STAT.F

The STAT.F script illustrates the structure and usage of a test script in the X.25 Monitor which collects frame and packet statistics. Test script function keys are defined to display statistics, display data, or clear the statistic counts.

```
( ---------------------------------------------------------------- )
( STAT.F       :  X.25 ANALYZER  Test Manager Scenario               )
(                 Collect and display X.25 statistics               )
(                 COUNTER1 to COUNTER30 collect counts.             )
(          f1 = Display Statistics.                                 )
(          f2 = Display Data.                                       )
(          f3 = Clear Statistic counts.                             )
( ---------------------------------------------------------------- )

TCLR                  ( CLEAR Test manager dictionary )
WAKEUP_ON             ( Wakeup the test manager to display statistics on startup )

                      ( Define sequence 4 to update and display I frame count )
4 SEQ{
      1 COUNTER4 +!                      ( Statistic updating for I frames )
      4 7 THERE  COUNTER4  @ W.          ( Display I frame count )
  }SEQ

0 STATE_INIT{
      " Show Stat" 1 LABEL_KEY
      " Show Data" 2 LABEL_KEY
      " Clear"     3 LABEL_KEY
      " Stop Test" 4 LABEL_KEY
  }STATE_INIT

0 STATE{           ( Define state 0 to collect statistics and ck events )
      R*SABM  1  ?RX                     ( SABM received )
      ACTION{
          OPEN_USER                      ( Open user display window )
          1 COUNTER1  +!                 ( Update SABM counter )
          1 7 THERE  COUNTER1  @ W.      ( Display SABM count )
          CLOSE_WINDOW                   ( Close display window )
      }ACTION
```

```
R*UA  1  ?RX                          ( UA received )
ACTION{
    OPEN_USER                         ( Open user display window )
    1 COUNTER2  +!                    ( Update UA counter)
    2 7 THERE  COUNTER2  @ W.         ( Display UA count )
    CLOSE_WINDOW                      ( Close display window )
}ACTION

R*DISC  1  ?RX                        ( DISC received )
ACTION{
    OPEN_USER                         ( Open user display window )
    1 COUNTER3  +!                    ( Update DISC counter )
    3 7 THERE  COUNTER3  @ W.         ( Display DISC count )
    CLOSE_WINDOW                      ( Close display window )
}ACTION

R*RR  1  ?RX                          ( RR frame received )
ACTION{
    OPEN_USER                         ( Open user display window )
    1 COUNTER5  +!                    ( Update RR frame counter )
    5 7 THERE  COUNTER5  @ W.         ( Display RR frame count )
    CLOSE_WINDOW                      ( Close display window )
}ACTION

R*RNR  1 ?RX                          ( RNR frame received )
ACTION{
    OPEN_USER                         ( Open user display window )
    1 COUNTER6  +!                    ( Update RNR frame counter )
    6 7 THERE  COUNTER6  @ W.         ( Display RNR frame count )
    CLOSE_WINDOW                      ( Close display window )
}ACTION

R*REJ  1  ?RX                         ( REJ frame received )
ACTION{
    OPEN_USER                         ( Open user display window )
    1 COUNTER7  +!                    ( Update REJ frame counter )
    7 7 THERE  COUNTER7  @ W.         ( Display REJ frame count )
    CLOSE_WINDOW                      ( Close display window )
}ACTION

R*DM  1  ?RX                          ( DM frame received )
ACTION{
    OPEN_USER                         ( Open user display window )
    1 COUNTER8  +!                    ( Update DM frame counter )
    8 7 THERE  COUNTER8  @ W.         ( Display DM frame count )
    CLOSE_WINDOW                      ( Close display window )
}ACTION
```

```
R*FRMR  1  ?RX                  ( FRMR frame received )
ACTION{
    OPEN_USER                   ( Open user display window )
    1 COUNTER9  +!              ( Update FRMR frame counter )
    9 7 THERE  COUNTER9  @ W.   ( Display FRMR frame count )
    CLOSE_WINDOW                ( Close display window )
}ACTION

R*INVFRM  1  ?RX                ( Invalid frame received )
ACTION{
    OPEN_USER                   ( Open user display window )
    1 COUNTER30  +!             ( Update invalid frame count )
    10 7 THERE  COUNTER30  @ W. ( Display invalid frame count )
    CLOSE_WINDOW                ( Close display window )
}ACTION

R*CALLREQ  1  ?RX               ( Call request received )
ACTION{
    OPEN_USER                   ( Open user display window )
    1 COUNTER10  +!             ( Update call request counter )
    1 48 THERE  COUNTER10  @ W. ( Display call request count )
    4 RUN_SEQ                   ( Display I frame count )
    CLOSE_WINDOW                ( Close display window )
}ACTION

R*CALLCON  1  ?RX               ( Call connect received )
ACTION{
    OPEN_USER                   ( Open user display window )
    1 COUNTER11  +!             ( Update call connect counter )
    2 48 THERE  COUNTER11  @ W. ( Display call connect count )
    4 RUN_SEQ                   ( Display I frame count )
    CLOSE_WINDOW                ( Close display window )
}ACTION

R*CLEARREQ  1  ?RX              ( Clear request received )
ACTION{
    OPEN_USER                   ( Open user display window )
    1 COUNTER12  +!             ( Update clear request counter )
    3 48 THERE  COUNTER12  @ W. ( Display clear request count )
    4 RUN_SEQ                   ( Display I frame count )
    CLOSE_WINDOW                ( Close display window )
}ACTION

R*CLEARCONF  1  ?RX             ( Clear confirm received )
ACTION{
    OPEN_USER                   ( Open user display window )
    1 COUNTER13  +!             ( Update clear confirm counter )
    4 48 THERE  COUNTER13  @ W. ( Display clear confirm count )
    4 RUN_SEQ                   ( Display I frame count )
    CLOSE_WINDOW                ( Close display window )
}ACTION
```

```
R*RESTARTREQ  1  ?RX              ( Restart request received )
ACTION{
    OPEN_USER                     ( Open user display window )
    1 COUNTER14  +!               ( Update restart request count )
    5 48 THERE  COUNTER14  @ W.   ( Display restart request count )
    4 RUN_SEQ                     ( Display I frame count )
    CLOSE_WINDOW                  ( Close display window )
}ACTION

R*RESTARTCONF  1  ?RX             ( Restart confirm received )
ACTION{
    OPEN_USER                     ( Open user display window )
    1 COUNTER15  +!               ( Update restart confirm count )
    6 48 THERE  COUNTER15  @ W.   ( Display restart confirm count )
    4 RUN_SEQ                     ( Display I frame count )
    CLOSE_WINDOW                  ( Close display window )
}ACTION

R*RESETREQ  1  ?RX                ( Reset request received )
ACTION{
    OPEN_USER                     ( Open user display window )
    1 COUNTER16  +!               ( Update reset request count )
    7 48 THERE  COUNTER16  @ W.   ( Display reset request count )
    4 RUN_SEQ                     ( Display I frame count )
    CLOSE_WINDOW                  ( Close display window )
}ACTION

R*RESETCONF  1  ?RX               ( Reset confirm received )
ACTION{
    OPEN_USER                     ( Open user display window )
    1 COUNTER17  +!               ( Update reset confirm count )
    8 48 THERE  COUNTER17  @ W.   ( Display reset confirm count )
    4 RUN_SEQ                     ( Display I frame count )
    CLOSE_WINDOW                  ( Close display window )
}ACTION

R*DATAP  1  ?RX                   ( Data packet received )
ACTION{
    OPEN_USER                     ( Open user display window )
    1 COUNTER18  +!               ( Update data packet count )
    9 48 THERE  COUNTER18  @ W.   ( Display data packet count )
    4 RUN_SEQ                     ( Display I frame count )
    CLOSE_WINDOW                  ( Close display window )
}ACTION
```

```
R*RRP  1  ?RX                       ( RR packet received )
ACTION{
    OPEN_USER                       ( Open user display window )
    1 COUNTER19  +!                 ( Update RR packet count )
    10 48 THERE COUNTER19  @ W.     ( Display RR packet count )
    4 RUN_SEQ                       ( Display I frame count )
    CLOSE_WINDOW                    ( Close display window )
}ACTION

UF1  ?KEY  ?WAKEUP OR               ( F1 key or TM_RUN wakeup events )
ACTION{                             ( Open and show user display window )
    POP_USER                        ( Open user display window )
    CLEAR_TEXT  WHI_FG PAINT        ( Clear screen text and color )
    13  0 THERE  W." f1    = Show Statistics, "
                 W." f2    = Show Data, "
                 W." f3    = Clear Statistics "
    1   0 THERE  W." SABM  = "  COUNTER1   @ W.    ( Display Statistics )
    2   0 THERE  W." UA    = "  COUNTER2   @ W.
    3   0 THERE  W." DISC  = "  COUNTER3   @ W.
    4   0 THERE  W." I     = "  COUNTER4   @ W.
    5   0 THERE  W." RR    = "  COUNTER5   @ W.
    6   0 THERE  W." RNR   = "  COUNTER6   @ W.
    7   0 THERE  W." REJ   = "  COUNTER7   @ W.
    8   0 THERE  W." DM    = "  COUNTER8   @ W.
    9   0 THERE  W." FRMR  = "  COUNTER9   @ W.
    10  0 THERE  W." INV   = "  COUNTER30  @ W.
    1  30 THERE  W." CALL REQUEST    = "  COUNTER10  @ W.
    2  30 THERE  W." CALL CONNECT    = "  COUNTER11  @ W.
    3  30 THERE  W." CLEAR REQUEST   = "  COUNTER12  @ W.
    4  30 THERE  W." CLEAR CONFIRM   = "  COUNTER13  @ W.
    5  30 THERE  W." RESTART REQUEST = "  COUNTER14  @ W.
    6  30 THERE  W." RESTART CONFIRM = "  COUNTER15  @ W.
    7  30 THERE  W." RESET REQUEST   = "  COUNTER16  @ W.
    8  30 THERE  W." RESET CONFIRM   = "  COUNTER17  @ W.
    9  30 THERE  W." DATA PACKET     = "  COUNTER18  @ W.
    10 30 THERE  W." RR PACKET       = "  COUNTER19  @ W.
    CLOSE_WINDOW
}ACTION

UF2  ?KEY
ACTION{
    SHOW_DATA                       ( Show data window )
}ACTION

UF3  ?KEY
ACTION{                             ( Clear statistic counters )
    0 COUNTER1  !  0 COUNTER2  !  0 COUNTER3  !  0 COUNTER4  !  0 COUNTER5  !
    0 COUNTER6  !  0 COUNTER7  !  0 COUNTER8  !  0 COUNTER9  !  0 COUNTER10 !
    0 COUNTER11 !  0 COUNTER12 !  0 COUNTER13 !  0 COUNTER14 !  0 COUNTER15 !
    0 COUNTER16 !  0 COUNTER17 !  0 COUNTER18 !  0 COUNTER19 !  0 COUNTER30 !
}ACTION
```

```
    UF4  ?KEY
    ACTION{
        TM_STOP  CLEAR_TEXT  WHI_FG PAINT
        " UF1"  1  LABEL_KEY
        " UF2"  2  LABEL_KEY
        " UF3"  3  LABEL_KEY
        " UF4"  4  LABEL-KEY
    }ACTION
}STATE                                  ( End of state 0 )
```

14.2 TEST_SEQ1.F

In this example, the tester is used to examine the operation of another device by transmitting a series of invalid control fields within frames. There are ten passes of this test and for each pass, an invalid frame is sent, first in the link disconnected state, and then in the link connected state. There should be no response from the partner in the link disconnected state. An FRMR is expected in the link operational state.

```
(-----------------------------------------------------------)
( Script file      :  TEST_SEQ1.F                           )
( Date Modified    :  August 12, 1988                       )
( Test reaction to invalid frames received by terminal.     )
( ENTER FUNCTION KEY  F1 TO START THE TEST                  )
( COUNTER IS USED AS THE LOOP COUNTER AND SERVES AS AN      )
( INDEX FOR IDENTIFYING THE INVALID FRAME TO BE SENT.       )
(-----------------------------------------------------------)

TCLR                                    ( INITIALIZE TEST MANAGER )
BLU_BG  TCOLOR                          ( Set trace report color to blue )

0 STATE_INIT{
      " START TEST"  1  LABEL_KEY
  }STATE_INIT

( Define command sequences )
0 SEQ{
      COUNTER @       ( Command Sequence 0 used to send invalid frames )
      DOCASE
          CASE 0       { X" 030D" }       ( Generate different invalid frames )
          CASE 1       { X" 0383" }       ( Depending on the value in COUNTER )
          CASE 2       { X" 03C3" }
          CASE 3       { X" 03A3" }
          CASE 4       { X" 03E3" }
          CASE 5       { X" 0307" }
          CASE 6       { X" 0347" }
          CASE 7       { X" 03E7" }
          CASE 8       { X" 038F" }
          CASE 9       { X" 03CF" }
          CASE 10
          ORCASE DUP   { X" 03AF" }       ( Default if COUNTER > 10 )
      ENDCASE
      SENDF                               ( Send and display the invalid frame )
  }SEQ

( STATE DEFINITIONS NOW FOLLOW )
```

```
0 STATE{                                ( Test Manager state 0 )
     UF1  ?KEY                          ( Start test when f1 key pressed )
     ACTION{
         T. " Test Starting"  TCR
         0 NEW_L2_STATE                 ( Turn emulation state machine off )
         0 COUNTER  !                   ( Zero loop counter )
         S:DISC                         ( Send DISC frame )
         1 NEW_STATE                    ( Enter TM state 1 )
     }ACTION

     R*SABM  1  ?RX                     ( Execute protocol state machine )
     ACTION{
     }ACTION
  }STATE

1 STATE{                                ( Test Manager state 1 - link is down )
     R*UA  R*DM  2  ?RX_FRAME           ( Check if UA or DM frames received )
     ACTION{
         0 RUN_SEQ                      ( Send next invalid frame type )
         START_T1                       ( Start T1 timer )
         2 NEW_STATE                    ( Enter TM state 2 )
     }ACTION

     T1-TIMER  ?TIMER                   ( Check if T1 timeout )
     ACTION{
         T. " TIMEOUT DETECTED IN STATE 1"   TCR
         TM_STOP  SABM                  ( Stop Test Manager and reset link )
     }ACTION

     R*SABM  1  ?RX                     ( Execute protocol state machine )
     ACTION{
     }ACTION
  }STATE

2 STATE{                                ( Test Manager State 2 )
     T1-TIMER  ?TIMER                   ( Check if T1 timeout )
     ACTION{
         S:SABM                         ( Send SABM frame )
         3 NEW_STATE                    ( Enter TM State 3 )
     }ACTION

     OTHER_EVENT
     ACTION{
         T." INVALID EVENT RECEIVED IN STATE 2"  TCR
         TM_STOP  SABM                  ( Stop Test manager and reset link )
     }ACTION
  }STATE
```

```
3 STATE{                                ( Test Manager State 3 - link is up )
    R*UA  1 ?RX_FRAME                   ( Check if UA frame received )
    ACTION{
        0 RUN_SEQ                       ( Send next invalid frame )
        START_T1                        ( Start T1 timer )
        4 NEW_STATE                     ( Enter TM State 4 )
    }ACTION
 }STATE

4 STATE{                                ( Test Manager State 4 )
    R*FRMR  1  ?RX_FRAME                ( Check if FRMR frame received )
    ACTION{
        STOP_T1                         ( Stop T1 timer )
        1 COUNTER +! COUNTER @ 11 =     ( Check if COUNTER is 11 )
        IF
            T." TEST FINISHED" TCR      ( If it is, test is finished )
            " UF1" 1 LABEL_KEY          ( Reset the testkey )
            TM_STOP  SABM               ( Stop Test Manager and reset link )
        ELSE
            S:DISC                      ( Otherwise send DISC frame )
            1 NEW_STATE                 ( Enter TM state 1 )
        THEN
    }ACTION

    OTHER_EVENT
    ACTION{
        T." INVALID EVENT RECEIVED IN STATE 4" TCR
        TM_STOP
        T." Test Finished." TCR
        " UF1" 1 LABEL_KEY
        SABM                            ( Stop Test Manager and reset link )
    }ACTION
 }STATE
```

14.3 TEST_SEQ2.F

In this example, calls are established on eight logical channels. These calls are subsequently cleared after eight call confirmations are received. These sequence is repeated three times.

```
(--------------------------------------------------------------------------------)
( Script File     :  TEST_SEQ2.F                                                  )
( Date Modified   :  August 12, 1988                                              )
( ENTER FUNCTION KEY F1 TO START THE TEST                                         )
( COUNTER IS USED TO COUNT THE CALL CONFIRMATION AND CLEAR CONFIRMATION           )
(      RESPONSES                                                                  )
( COUNTER1 IS USED AS THE LOOP COUNTER.  TO MODIFY THE NUMBER OF PASSES           )
(      CHANGE THE MAXIMUM VALUE IN STATE 5                                        )
(--------------------------------------------------------------------------------)

TCLR
0 STATE_INIT{
      " START TEST"
      1 LABEL_KEY
  }STATE_INIT

0 STATE{
      UF1  ?KEY                         ( Wait for UF1 function key )
      ACTION{
          T." Test Starting"  TCR
          0 COUNTER1  !                 ( Start with pass zero )
          SABM                          ( Establish link )
          1 NEW_STATE
      }ACTION

      R*SABM  1  ?RX                    ( Execute protocol state machine )
      ACTION{
      }ACTION
  }STATE

1 STATE{
      R*UA  1  ?RX                      ( Layer 2 active )
      ACTION{
          RESTART                       ( Send restart packet )
          2  NEW_STATE
      }ACTION
  }STATE
```

```
2 STATE{
    R*RESTARTCONF  1  ?RX
    ACTION{
        CH1  CALL                       ( Send calls on eight logical channels )
        CH2  CALL
        CH3  CALL
        CH4  CALL
        CH5  CALL
        CH6  CALL
        CH7  CALL
        CH8  CALL
        0 COUNTER  !                    ( Zero the call-counter )
        3 NEW_STATE
    }ACTION
}STATE

3 STATE{
    R*CALLCON  1  ?RX
    ACTION{
        1 COUNTER  +!                   ( Increment for each call connect )
        COUNTER  @ 8 =
        IF
            CH1  CLEAR                  ( Clear all LCN's after 8 connects )
            CH2  CLEAR
            CH3  CLEAR
            CH4  CLEAR
            CH5  CLEAR
            CH6  CLEAR
            CH7  CLEAR
            CH8  CLEAR
            0 COUNTER  !                ( Reset the call-counter )
            4 NEW_STATE
        ENDIF
    }ACTION
}STATE

4 STATE{
    R*CLEARCCONF  1  ?RX                ( Wait for clear confirm packets )
    ACTION{
        1 COUNTER  +!
        COUNTER  @ 8 =
        IF
            DISC                        ( Send DISC after 8 clear confirm packets )
            5 NEW_STATE
        ENDIF
    }ACTION
}STATE
```

```
5 STATE{
      R*UA  1  ?RX
      ACTION{
          1 COUNTER1  +!                    ( Increment pass counter )
          COUNTER1  @ 3 =
          IF                                ( Stop if equal to three )
              T." Test Finished"  TCR
              TM_STOP
              " UF1"  1 LABEL_KEY
          ELSE                              ( Otherwise repeat )
              SABM
              1 NEW_STATE
          ENDIF
      }ACTION
  }STATE
```

14.4 TEST_SEQ3.F

In this example, calls are established on eight logical channels. A total of 100 data packets are transmitted, following which, the calls are cleared. The entire sequence is repeated three times.

```
(----------------------------------------------------------------------------)
( Script File    :  TEST_SEQ3.F                                               )
( Date Modified  :  August 12, 1988                                           )
( ENTER FUNCTION KEY F1 TO START THE TEST                                     )
( COUNTER IS USED TO COUNT THE CALL CONFIRMATION AND CLEAR CONFIRMATION       )
(       RESPONSES                                                             )
( COUNTER1 IS USED TO COUNT THE TRANSMITTED DATA PACKETS, THE MAXIMUM         )
(       VALUE CAN BE MODIFIED IN STATE 4 AND 5                                )
( COUNTER2 IS USED TO COUNT THE RECEIVED RR PACKETS.  THE NUMBER MUST         )
(       BE MATCHED TO THE NUMBER OF TRANSMITTED DATA PACKETS                  )
( COUNTER3 IS THE LOOP COUNTER.  THE NUMBER OF PASSES CAN BE MODIFIED         )
(       IN STATE 7                                                            )
(----------------------------------------------------------------------------)

TCLR
0 STATE_INIT{
      " START KEY"                       ( Label Testkey UF1 as START KEY )
      1 LABEL_KEY
  }STATE_INIT

0 STATE{
      UF1  ?KEY                          ( Has Testkey UF1 bee passed? )
      ACTION{
          T." Test Starting"  TCR        ( Create trace statement )
          0 COUNTER3  !                  ( Initialize loop counter )
          SABM                           ( Send SABM )
          1 NEW_STATE
      }ACTION

      R*SABM  1  ?RX                     ( Execute protocol state machine )
      ACTION{
      }ACTION
  }STATE

1 STATE{
      R*UA  1  ?RX                       ( UA frame received? )
      ACTION{
          RESTART                        ( Send Restart packet )
          2 NEW_STATE
      }ACTION
  }STATE
```

```
2 STATE{
     R*RESTARTCONF  1  ?RX            ( Restart confirmation packet received? )
     ACTION{
         CH1  CALL                    ( Establish 8 calls )
         CH2  CALL
         CH3  CALL
         CH4  CALL
         CH5  CALL
         CH6  CALL
         CH7  CALL
         CH8  CALL
         0 COUNTER  !                 ( Zero the call-counter )
         3 NEW_STATE                  ( Go to State 3 )
     }ACTION
  }STATE

3 STATE{
     R*CALLCON  1  ?RX                ( Call connect packet received? )
     ACTION{
         1 COUNTER  +!                ( Increment call-counter )
         COUNTER  @ 8 =               ( Have 8 call connects been received? )
         IF                           ( Yes )
             CH1  DATA DATA           ( Send 2 data packets on each channel )
             CH2  DATA DATA
             CH3  DATA DATA
             CH4  DATA DATA
             CH5  DATA DATA
             CH6  DATA DATA
             CH7  DATA DATA
             CH8  DATA DATA
             16 COUNTER1  !           ( Increase data packet counter by 16 )
             0  COUNTER2  !           ( Initialize RR packet counter to 0 )
             4  NEW_STATE             ( Go to State 4 )
         ENDIF
     }ACTION
  }STATE

4 STATE{
     R*RRP  1  ?RX                    ( RR packet received? )
     ACTION{
         DATA                         ( Send a data packet on same channel )
         1 COUNTER2  +!               ( Increment RR packet counter )
         1 COUNTER1  +!               ( Increment data packet counter )
         COUNTER1  @ 100 =            ( Have 100 data packets been sent? )
         IF                           ( Yes )
             5 NEW_STATE              ( Go to State 5 )
         ENDIF
     }ACTION
  }STATE
```

```
5 STATE{
     R*RRP  1  ?RX                      ( RR packet received? )
     ACTION{
          1 COUNTER2  +!                ( Increment RR counter )
          COUNTER2  @ 100 =             ( Have 100 RR packets been received? )
          IF                            ( Yes )
               CH1  CLEAR               ( Clear the eight logical channels )
               CH2  CLEAR
               CH3  CLEAR
               CH4  CLEAR
               CH5  CLEAR
               CH6  CLEAR
               CH7  CLEAR
               CH8  CLEAR
               0 COUNTER  !
               6 NEW_STATE              ( Go to State 6 )
          ENDIF
     }ACTION
  }STATE

6 STATE{
     R*CLEARCONF  1  ?RX                ( Clear confirmation packet received? )
     ACTION{
          1 COUNTER  +!                 ( Increment clear counter )
          COUNTER  @ 8 =                ( Have 8 clear confirm packets been received? )
          IF                            ( Yes )
               DISC                     ( Send a disconnect )
               7 NEW_STATE              ( Go to State 7 )
          ENDIF
     }ACTION
  }STATE

7 STATE{
     R*UA  1  ?RX                       ( Has UA frame been received? )
     ACTION{
          1 COUNTER3  +!                ( Increment loop counter )
          COUNTER3  @ 3 =               ( Has test been repeated 3 times? )
          IF                            ( Yes )
               T." Test Finished"  TCR  ( Create trace statement )
               TM_STOP                  ( Stop Test Manager )
               " UF1"  1 LABEL_KEY      ( Relabel testkey to UF1 )
          ELSE                          ( No )
               SABM                     ( Send SABM )
               1 NEW_STATE              ( Report test )
          ENDIF
     }ACTION
  }STATE
```

14.5 TEST_SEQ4.F

This example generates calls on eight logical channels. Two data packets are then generated on each logical channel with additional data packets sent as RR packet acknowledgements are received. The test is started by pressing the test script *START TEST* function key (UF1) and can be stopped by pressing the test script *STOP TEST* function key (UF2).

NOTE
The number of data packets actually generated in state 3 of this script is dependent on the window size selected for packets on the Emulation Configuration Menu. In this example, if the window size had been set to 1, only one data packet would be generated in state 3 for each channel. The second data packet would be discarded.

This script can be altered to generate more data packets. If the packet window size has been set to 7, seven data packets could be generated by changing the script in state 3 i.e. to CH1 DATA DATA DATA DATA DATA DATA DATA etc. for each channel. See Section 9 for information on the = WINDOW and K commands.

```
(--------------------------------------------------------------------)
( Script File     :  TEST_SEQ4.F                                      )
( Data Modified   :  August 12, 1988                                  )
( ENTER FUNCTION KEY F1 TO START THE TEST, FUNCTION KEY F2 TO         )
(       STOP THE TEST                                                 )
( COUNTER IS USED TO COUNT THE CALL CONFIRMATION RESPONSES            )
(--------------------------------------------------------------------)

TCLR
0 STATE_INIT{
      " START TEST"  1 LABEL_KEY         ( Label Testkey UF1 as Start Test )
      " STOP TEST"   2 LABEL_KEY         ( Label Testkey UF2 as Stop Test )
  }STATE_INIT

0 STATE{
      UF1  ?KEY                          ( Has Testkey UF1 been pressed? )
      ACTION{
          T." Test Starting"  TCR        ( Created trace statement )
          SABM                           ( Send SABM )
          1 NEW_STATE
      }ACTION

      R*SABM  1  ?RX                     ( Execute protocol state machine)
      ACTION{
      }ACTION
  }STATE
```

```
1 STATE{
     R*UA  1  ?RX                        ( UA frame received? )
     ACTION{
         RESTART                         ( Send restart packet )
         2 NEW_STATE
     }ACTION
  }STATE

2 STATE{
     R*RESTARTCONF  1  ?RX               ( Restart confirmation packet received? )
     ACTION{
         CH1  CALL                       ( Send out call request on 8 channels )
         CH2  CALL
         CH3  CALL
         CH4  CALL
         CH5  CALL
         CH6  CALL
         CH7  CALL
         CH8  CALL
         0 COUNTER  !                    ( Initialize call confirm counter )
         3 NEW_STATE                     ( Go to State 3 )
     }ACTION
  }STATE

3 STATE{
     R*CALLCON  1  ?RX                   ( Call confirm packet received? )
     ACTION{
         1 COUNTER  +!                   ( Increment call confirm counter )
         COUNTER  @ 8 =                  ( Have 8 call confirms been received? )
         IF                              ( Yes )
             CH1  DATA DATA              ( Send 2 data packets on each channel )
             CH2  DATA DATA
             CH3  DATA DATA
             CH4  DATA DATA
             CH5  DATA DATA
             CH6  DATA DATA
             CH7  DATA DATA
             CH8  DATA DATA
             4 NEW_STATE                 ( Go to State 4 )
         ENDIF
     }ACTION
  }STATE
```

```
4 STATE{
     R*RRP  1  ?RX                         ( RR packet received? )
     ACTION{
         DATA                              ( Send a data packet on same channel )
     }ACTION

     UF2  ?KEY                             ( Has testkey UF2 been pressed? )
     ACTION{
         RESTART                           ( Send a restart packet )
         5 NEW_STATE                       ( Go to State 5 )
     }ACTION
  }STATE

5 STATE{
     R*RESTARTCONF  1  ?RX                 ( Has restart confirm packet been received? )
     ACTION{
         DISC                              ( Send a disconnect frame )
         6 NEW_STATE                       ( Go to State 6 )
     }ACTION
  }STATE

6 STATE{
     R*UA  1  ?RX                          ( Has UA frame been received? )
     ACTION{
         T." Test Finished"  TCR           ( Create trace statement )
         TM_STOP                           ( Stop Test Manager )
         " UF1"  1 LABEL_KEY               ( Relabel testkeys to UF1 and UF2 )
         " UF2"  2 LABEL_KEY
     }ACTION
  }STATE
```

14.6 TEST_SEQ5.F

In this example, calls are established and cleared continuously on eight logical channels. A timer allows the test to run for 100 seconds, at which point the total number of calls is reported.

```
(-----------------------------------------------------------------)
( Script File     :  TEST_SEQ5.F                                  )
( Date Modified   :  August 12, 1988                              )
( ENTER FUNCTION KEY F1 TO START THE TEST                         )
( COUNTER CONTAINS THE NUMBER OF CALL REQUESTS/INDICATIONS        )
(-----------------------------------------------------------------)

TCLR
0 STATE_INIT{
      " START TEST"  1 LABEL_KEY        ( Label testkey UF1 as Start Test )
  }STATE_INIT

0 STATE{
      UF1  ?KEY                         ( Has testkey UF1 been pressed? )
      ACTION{
          T." TEST STARTING"  TCR       ( Create trace statement )
          SABM                          ( Send SABM )
          1 NEW_STATE                   ( Go to State 1 )
      }ACTION
  }STATE

1 STATE{
      R*UA  1  ?RX                      ( Has UA frame been received? )
      ACTION{
          RESTART                       ( Send restart request packet )
          2 NEW_STATE
      }ACTION

      R*SABM  1  ?RX                    ( Execute protocol state machine )
      ACTION{
      }ACTION
  }STATE
```

```
2 STATE{
     R*RESTARTCONF  1  ?RX            ( Has restart confirm packet been received? )
     ACTION{
         CH1  CALL                    ( Send a call request on 8 channels )
         CH2  CALL
         CH3  CALL
         CH4  CALL
         CH5  CALL
         CH6  CALL
         CH7  CALL
         CH8  CALL
         8 COUNTER  !                 ( Store 8 in call counter )
         20  1000 START_TIMER         ( Start timer 20 )
         3 NEW_STATE                  ( Go to State 3 )
     }ACTION
  }STATE

3 STATE{
     R*CALLCON  1  ?RX                ( Has call accept been received? )
     ACTION{
         CLEAR                        ( Send a clear request packet )
     }ACTION

     R*CLEARCONF  1  ?RX              ( Has clear Confirm been received? )
     ACTION{
         CALL                         ( Send a call request on same channel )
         1 COUNTER  +!                ( Increment call counter )
     }ACTION

     20 ?TIMER                        ( Any time expired? )
     ACTION{
         RESTART                      ( Send restart packet )
         4 NEW_STATE                  ( Go to State 4 )
     }ACTION
  }STATE

4 STATE{
     R*RESTARTCONF  1  ?RX            ( Has restart confirm been received? )
     ACTION{
         DISC                         ( Send a disconnect frame )
         5 NEW_STATE                  ( Go to State 5 )
     }ACTION
  }STATE
```

```
5 STATE{
     R*UA  1  ?RX
     ACTION{
         T." TEST FINISHED"  TCR          ( Create trace statement )
         COUNTER @ .
         " CALLS IN 100 SECONDS"
         BLU_BG DISPLAY
         TM_STOP                          ( Stop Test Manager )
         " UF1"  1 LABEL_KEY              ( Relabel testkey 1 )
     }ACTION
  }STATE
```

14.7 TEST_SEQ6.F

In this example, the manipulation of the LCN variable is illustrated. In state 2, a user programmed call request has been encoded and is transmitted using the SENDP command. The LCN state is forced to STATE 2 – Calling. Likewise, in state 3, a clear request packet is manually generated, and the LCN state is forced to 6 – Clearing.

NOTE
The value 1001 has been entered in the GFI_LCN field. The user must ensure that this logical channel number has been assigned to CH1 on the LCN Setup Menu.

```
(----------------------------------------------------)
( Script File    :  TEST_SEQ6.F                      )
( Date Modified  :  August 12, 1988                  )
( ENTER FUNCTION KEY CF1 TO START THE TEST           )
(----------------------------------------------------)

TCLR
0 STATE_INIT{
      " START TEST"  1 LABEL_KEY        ( Label testkey UF1 as Start Test )
  }STATE_INIT

0 STATE{
      UF1  ?KEY                         ( Has testkey UF1 been pressed? )
      ACTION{
          T." Test Starting"  TCR       ( Create trace statement )
          0 COUNTER1  !                 ( Initialize counter )
          SABM                          ( Send SABM )
          1 NEW_STATE                   ( Go to State 1 )
      }ACTION

      R*SABM  1  ?RX                    ( Execute protocol state machine )
      ACTION{
      }ACTION
  }STATE

1 STATE{
      R*UA  1  ?RX                      ( Has UA frame been received? )
      ACTION{
          RESTART                       ( Send restart packet )
          2 NEW_STATE                   ( Go to State 2 )
      }ACTION
  }STATE
```

```
2 STATE{
      R*RESTARTCONF  1  ?RX            ( Has restart confirm been received? )
      ACTION{
          CH1  X" 10010B22303000" SENDP ( Manually transmit a call request )
          2 NEW_LCN_STATE              ( Change LCN state to calling )
          3 NEW_STATE                  ( Go to State 3 )
      }ACTION
  }STATE

3 STATE{
      R*CALLCON  1  ?RX                ( Has call connect been received? )
      ACTION{
          X" 1001130000" SENDP         ( Manually transmit a clear request )
          6 NEW_LCN_STATE              ( Change LCN state to clearing )
          4 NEW_STATE                  ( Go to State 4 )
      }ACTION
  }STATE

4 STATE{
      R*CLEARCONF  1  ?RX              ( Has clear confirm been received? )
      ACTION{
          DISC                         ( Send disconnect )
          5 NEW_STATE                  ( Go to State 5 )
      }ACTION

5 STATE{
      R*UA  1  ?RX                     ( Has UA frame been received? )
      ACTION{
          T." Test Finished"  TCR      ( Create trace statement )
          TM_STOP                      ( Stop Test Manager )
          " UF1"  1 LABEL_KEY          ( Relabel testkey )
      }ACTION
  }STATE
```

14.8 TEST_SEQ7.F

This example illustrates the use of subroutines, i.e. LOAD_RETURN_STATE and RETURN_STATE as well as an example of the CASE construct. Calls are established on four logical channels; twenty data packets are transmitted; and then, all channels are cleared.

The subroutine (states 250 through 252) perform the repetitive operations.

```
(-----------------------------------------------------------------)
( Test Script     :  TEST_SEQ7.F                                  )
( Date Modified   :  August 12, 1988                              )
( ENTER FUNCTION     KEY F1 TO START THE TEST                     )
( COUNTER            COUNTS THE CALL SEQUENCES                    )
( COUNTER1           COUNTS THE TRANSMITTED DATA PACKETS          )
( COUNTER2           COUNTS THE RECEIVED RR PACKETS               )
(-----------------------------------------------------------------)

TCLR
0 STATE_INIT{
      " START TEST"  1 LABEL_KEY        ( Label testkey UF1 as Start Test )
  }STATE_INIT

0 STATE{
      UF1  ?KEY                         ( Has UF1 key been pressed? )
      ACTION{
          T." Test Starting"  TCR       ( Create trace statement )
          0 COUNTER !                   ( Initialize clear confirm counter )
          SABM                          ( Send SABM )
          1 NEW_STATE                   ( Go to State 1 )
      }ACTION

      R*SABM  1  ?RX                    ( Execute protocol state machine )
      ACTION{
      }ACTION
  }STATE

1 STATE{
      R*UA  1  ?RX                      ( Has UA been received? )
      ACTION{
          RESTART                       ( Send restart request packet )
          2 NEW_STATE                   ( Go to State 2 )
      }ACTION
  }STATE
```

```
2 STATE{
    R*RESTARTCONF  1  ?RX          ( Has restart confirm packet been received? )
    ACTION{
        CH1  CALL                  ( Send a call request on channel 1 )
        3 LOAD_RETURN_STATE        ( Set state number for return from )
                                   ( subroutine )
        250 NEW_STATE              ( Go to State 250 )
    }ACTION
  }STATE

3 STATE{
    R*CLEARCONF  1  ?RX            ( Has clear confirm packet been received? )
    ACTION{
        1 COUNTER  +!              ( Increment clear confirm Counter )
        COUNTER  @
        DOCASE                     ( Select channel for call and state number )
            CASE 1    {  CH2 3 }   ( for return from subroutine )
            CASE 2    {  CH3 3 }
            CASE 3    {  CH4 4 }
            CASE DUP  {      0 }
        ENDCASE
        LOAD_RETURN_STATE
        CALL                       ( Send call request on selected channel )
        250_NEW_STATE              ( Go to State 250 )
    }ACTION
  }STATE

4 STATE{
    R*CLEARCONF  1  ?RX            ( Has clear confirm packet been received? )
    ACTION{
        DISC                       ( Send disconnect )
        5 NEW_STATE                ( Go to State 5 )
    }ACTION
  }STATE

5 STATE{
    R*UA  1  ?RX                   ( Has UA been received? )
    ACTION{
        T." Test Finished"  TCR    ( Create trace statement )
        TM_STOP                    ( Stop Test Manager )
        " UF1"  1 LABEL_KEY        ( Relabel testkey )
    }ACTION
  }STATE
```

```
( THE SUBROUTINE SECTION )

250 STATE{
      R*CALLCON  1  ?RX              ( Has call connect packet been received? )
      ACTION{
          DATA DATA                  ( Send 2 data packets on this channel )
          2 COUNTER1  !              ( Store 2 in data packet counter )
          0 COUNTER2  !              ( Initialize RR packet counter )
          251 NEW_STATE              ( Go to State 251 )
      }ACTION
    }STATE

251 STATE{
      R*RRP  1  ?RX                  ( Has RR packet been received? )
      ACTION{
          DATA                       ( Send data packet )
          1 COUNTER2  +!             ( Increment RR packet counter )
          1 COUNTER1  +!             ( Increment data packet counter )
          COUNTER1  @ 20 =           ( Have we sent 20 data packets? )
          IF                         ( Yes )
              252 NEW_STATE          ( Go to State 252 )
          THEN
      }ACTION
    }STATE

252 STATE{
      R*RRP  1  ?RX                  ( Has RR packet been received? )
      ACTION{
          1 COUNTER2  +!             ( Increment RR packet counter )
          COUNTER 2  @ 20 =          ( Have we received 20 RR packets? )
          IF                         ( Yes )
              CLEAR RETURN_STATE     ( Send a clear request and return )
                                     ( from subroutine )
          THEN
      }ACTION
    }STATE
```

A
INTRODUCTION TO CCITT X.25

This appendix contains a brief introduction to X.25. For further information, the reader is advised to consult the CCITT X.25 (1984) Recommendation, Red Book VIII.3, Malaga-Torremolinos.

A.1 Purpose of the Recommendation

This recommendation was established to produce standards to facilitate international networking between various countries which have established public data networks providing packet switched data transmission services.

A.2 Types of X.25 Services

PVC (permanent virtual circuit) is the establishment of a permanent relationship between two users.

SVC (switched virtual circuit) is a relationship that is established only for the duration of a call. SVC requires a call setup phase, the delivery of data in sequences, and a call clearing phase to terminate the call.

A.3 Layer 1 – Physical Layer

The physical layer provides physical, mechanical, and electrical conditions for the synchronous transmission of the bit streams generated by higher protocol layers. This layer is specified by reference to interface recommendations. CCITT recommends the use of the X.21 interface but many network implementations utilize V.28/RS-232C, V.36/RS-449 or V.35 for high speed transmissions. All of these physical interfaces are supported on IDACOM testers.

A.4 Layer 2 – Link Layer

The link layer provides reliable point to point communications between a terminal and the network. Information is transferred across the link in frames. The access procedure used is HDLC (high level data link control) and mechanisms are provided to:

- initialize the link;
- ensure that any message can be transferred (transparency);
- minimize the probability of undetected errors;
- ensure that the information is transferred across the link in the correct sequence;
- control the flow of information across the link;
- detect and report procedure errors; and
- logically disconnect the link.

Frame Structure

The frame structure discussed here is LAPB described in the CCITT X.25 (1984) Recommendation. Frame sequence numbers, N(R) and N(S), cycle 0 through 7 for modulo 8, and 0 through 127 for modulo 128 .

All HDLC frames contain:

- a start flag;
- an address field;
- a control field;
- an information field (optional); and
- a closing flag.

Bit order of transmission	12345678	12345678	12345678	Optional	16 to 1	12345678
	Flag	Address	Control	Information	FCS	Flag
	F 01111110	A 8-bits	C 8-bits	Info N-bits	FCS 16-bits	F 01111110

Table A-1 Frame Structure – Modulo 8

Bit order of transmission	12345678	12345678	1 to *)	Optional	16 to 1	12345678
	Flag	Address	Control	Information	FCS	Flag
	F 01111110	A 8–bits	C *)–bits	Info N–bits	FCS 16–bits	F 01111110

*) 16 for frame formats containing sequence numbers; 8 for frame formats not containing sequence numbers

Table A–2 Frame Structure – Modulo 128

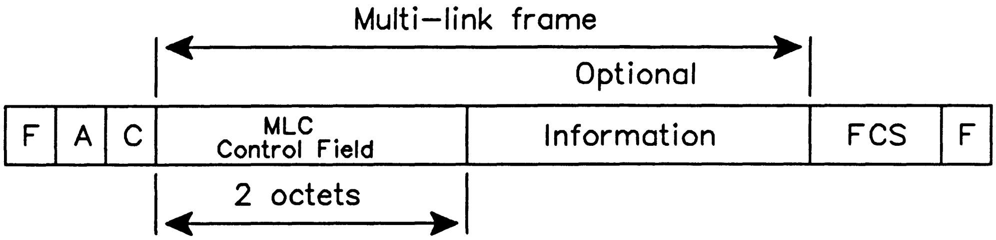


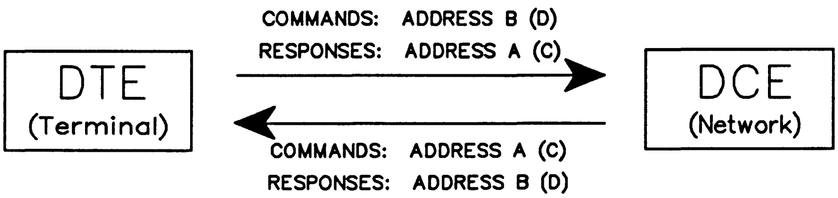
Figure A–1 Multilink Frame Formats

Flags

The flag prior to the address field is the opening flag, and the one following the FCS (frame check sequence) field is the closing flag. A single flag can be used as both the closing flag for one frame and the opening flag for the next frame. The bit pattern for these flags in X.25 is 0111 1110 (hex 7E).

Address

The address field consists of one octet. It identifies the intended receiver of a command frame and the transmitter of a response frame. These addresses are referred to as Address A with a value of 3, and Address B with a value of 1 for single link; Address C with a value of hex 15 and Address D with a value of 7 for multilink. Figure A–2 shows the addresses used for communications between a DTE and a DCE.



ADDRESS A = 3 ADDRESS C = 15
ADDRESS B = 1 ADDRESS D = 7

Figure A–2 X.25 MLP Address Field

NOTE
Single link procedure uses Addresses A and B; multilink procedure uses Addresses C and D.

FCS (Frame Check Sequence Field)

The FCS is a 16 bit field which is calculated at both ends of the link. If the receiver's calculated FCS does not equal the value sent in the frame, the transmission is incorrect and error recovery procedures are initiated.

Control Field

The control field identifies the type of frame and provides control information relevant to each type. The three types of frames are information, supervisory, and unnumbered.

Information frames:

- can only be commands;
- control byte contains:
  - a N(S) (send sequence number)
  - a N(R) (receive sequence number) – an acknowledgement of received frames
  - a P (poll) bit used to request confirmation; and
- contain additional octets containing packet information.

Supervisory frames:

- can be commands or responses;
- control field contains:
  - an N(R) (receive sequence number)
  - a P/F (poll/final) bit used to request confirmation; and
- include:
  - RR (receiver ready)
  - RNR (receiver not ready)
  - REJ (reject)

Unnumbered frames:

- contain information used for link control, establishing or disconnecting a link;
- control field contains:
  - a P/F (poll/final) bit used to request confirmation; and
- include:
  - SABM (set asynchronous balanced mode) for modulo 8
  - SABME (set asynchronous balanced mode extended) for modulo 128
  - DISC (disconnect)
  - DM (disconnected mode)
  - UA (unnumbered acknowledgement)
  - FRMR (frame reject)

Table A-3 shows the bit contents of the control fields for the previous frames for modulo 8.

Format	Command	Response	Encoding							
		Bits	1 (MSB)	2	3	4	5	6	7	8 (LSB)
Information transfer	I (information)		0	N(S)			P	N(R)		
Supervisory	RR (receive ready)	RR (receive ready)	1	0	0	0	P/F	N(R)		
	RNR (receive not ready)	RNR (receive not ready)	1	0	1	0	P/F	N(R)		
	REJ (reject)	REJ (reject)	1	0	0	1	P/F	N(R)		
Unnumbered	SABM (set asynchronous balanced mode)		1	1	1	1	P	1	0	0
	DISC (disconnect)		1	1	0	0	P	0	1	0
		DM (disconnected mode)	1	1	1	1	F	1	1	0
		UA (unnumbered acknowledgement)	1	1	0	0	F	1	1	0
		FRMR (frame reject)	1	1	1	0	F	0	0	1

Table A–3 LAPB Frame Control Fields – Modulo 8

Table A-4 shows the bit contents of the control fields for the previous frames for modulo 128.

Format	Command	Response	Encoding									
		Bits	1 (MSB)	2	3	4	5	6	7	8 (LSB)	9	10 to 16
Information transfer	I (information)		0	N(S)							P	N(R)
Supervisory	RR (receive ready)	RR (receive ready)	1	0	0	0	0	0	0	0	P/F	N(R)
	RNR (receive not ready)	RNR (receive not ready)	1	0	1	0	0	0	0	0	P/F	N(R)
	REJ (reject)	REJ (reject)	1	0	0	1	0	0	0	0	P/F	N(R)
Unnumbered	SABME (set asynchronous balanced mode extended)		1	1	1	1	P	1	1	0		
	DISC (disconnect)		1	1	0	0	P	0	1	0		
		DM (disconnected mode)	1	1	1	1	F	0	0	0		
		UA (unnumbered acknowledge-ment)	1	1	0	0	F	1	1	0		
		FRMR (frame reject)	1	1	1	0	F	0	0	1		

Table A-4 Frame Control Fields – Modulo 128

A.5 Layer 3 – Network Level

The network layer performs basic multiplexing of data and allows many different virtual circuits to be operated over a single link layer. It is at this level that call setup and clearing occurs, as well as access to network provided facilities. Data transfer is accomplished with a mechanism that ensures that information is transferred in the correct sequence.

The network layer information is contained in packets which are carried in information frames. The packet information commences in the first byte following the control field of an I frame. This byte is commonly called octet one when referring to packet formats.

The packet structure discussed here is for layer 3 modulo 8, only as described in the 1984 Recommendation. The user should consult the X.25 (1980/1984) Recommendation for modulo 128 structures.

Each packet contains a packet control header consisting of three octets. Table A-5 shows the format for the packet control header with the size of each field in bits.

Octets	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1
1	General format identifier				Logical channel group number			
2	Logical channel number							
3	Packet type identifier							

Table A–5 Packet Header

GFI (General Format Identifier)

This field identifies the general format identifier of the data packet sequencing scheme, i.e. modulo 8 or modulo 128. Packet sequence numbers cycle between 0 through 7 for modulo 8, and 0 through 127 for modulo 128. The modulo is determined by examining bits 5 and 6 of the first octet of the packet.

Data packets contain a Q bit within the GFI. This is the eighth bit of octet 1 in the data packet header. When the Q (qualifier) bit is set to 1, the data packet being sent is not user data.

Data packets and call setup packets contain a D bit. This is the seventh bit of octet 1 in the packet header. When the D (delivery confirmation) bit is set, an acknowledgement from a distant subscriber is required.

Logical Channel Group Number and Logical Channel Number

These fields show information regarding the destination of a packet over a logical virtual channel in the network.

Packet Identifier

Each packet is identified in octet 3 according to Table A-6. A bit which is marked as X in this table can be set to a 0 or 1.

Packet Type		Octet 3 Bits							
From DCE to DTE	From DTE to DCE	8	7	6	5	4	3	2	1
Call setup and clearing									
Incoming call	Call request	0	0	0	0	1	0	1	1
Call connected	Call accepted	0	0	0	0	1	1	1	1
Clear indication	Clear request	0	0	0	1	0	0	1	1
DCE clear confirmation	DTE clear confirmation	0	0	0	1	0	1	1	1
Data and interrupt									
DCE data	DTE data	X	X	X	X	X	X	X	0
DCE interrupt	DTE interrupt	0	0	1	0	0	0	1	1
DCE interrupt confirmation	DTE interrupt confirmation	0	0	1	0	0	1	1	1
Flow control and reset									
DCE RR (modulo 8)	DTE RR (modulo 8)	X	X	X	0	0	0	0	1
DCE RR (modulo 128)	DTE RR (modulo 128)	0	0	0	0	0	0	0	1
DCE RNR (modulo 8)	DTE RNR (modulo 8)	X	X	X	0	0	1	0	1
DCE RNR (modulo 128)	DTE RNR (modulo 128)	0	0	0	0	0	1	0	1
	DTE REJ (modulo 8)	X	X	X	0	1	0	0	1
	DTE REJ (modulo 128)	0	0	0	0	1	0	0	1
Reset indication	Reset request	0	0	0	1	1	0	1	1
DCE reset confirmation	DTE reset confirmation	0	0	0	1	1	1	1	1
Restart									
Restart indication	Restart request	1	1	1	1	1	0	1	1
DCE restart confirmation	DTE restart confirmation	1	1	1	1	1	1	1	1
Diagnostic									
Diagnostic		1	1	1	1	0	0	0	1
Registration									
	Registration request	1	1	1	1	0	0	1	1
Registration confirmation		1	1	1	1	0	1	1	1

Table A–6 Packet Type Identifier

NOTE
For Modulo 8, bits 6, 7, and 8 in RR, RNR, REJ, or data packets contain the packet receive sequence number P(R). Bits 2, 3, and 4 in data packets contain the packet send sequence number P(S). Bit 5 in data packets contains the M (more) bit. If this bit is set to 1, more data is to follow.

A.6 Example of Link Setup, Call Setup, Data Transfer, and Call Clear

Figure A–3 is an example of the commands and expected responses for link setup, call setup, data transfer, and call clear for an SVC.

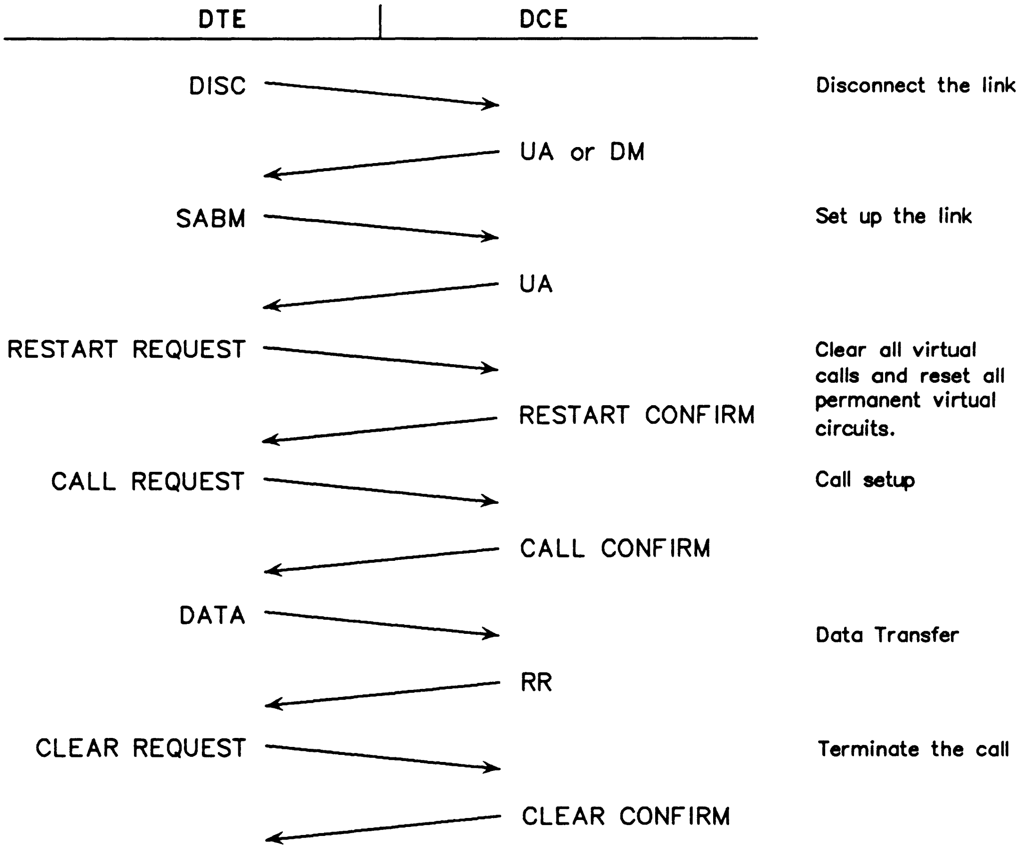


Figure A–3 Link Setup, Call Setup, Data Transfer, and Call Clearing

B
COMMAND SUMMARIES

Physical Events		
Command	Stack Description	Description
?CRC_ERROR	(--flag)	Detects a frame with a CRC error
?ABORT	(--flag)	Detects an abort on the line
?RTS_ON, ?RTS_OFF	(--flag)	Detects a transition on the request to send lead
?CTS_ON, ?CTS_OFF	(--flag)	Detects a transition on the clear to send lead
?DSR_ON, ?DSR_ON	(--flag)	Detects a transition on the data set ready lead
?CD_ON, ?CD_OFF	(--flag)	Detects a transition on the carrier detect lead
?DTR_ON, ?DTR_OFF	(--flag)	Detects a transition on the data terminal ready lead
?SQ_ON, ?SQ_OFF	(--flag)	Detects a transition on the signal quality lead
?RI_ON, ?RI_OFF	(--flag)	Detects a transition on the ring indication lead

Table B–1 Physical Events

Setting Leads		
Command	Stack Description	Description
RTS_ON , RTS_OFF	(--)	Sets the request to send lead (V.28, V.35, V.36)
CTS_ON, CTS_OFF	(--)	Sets the clear to send lead (V.28, V.35, V.36)
DSR_ON, DSR_ON	(--)	Sets the data set ready lead (V.28, V.35, V.36)
CD_ON, CD_OFF	(--)	Sets the carrier detect lead (V.28, V.35) and sets the channel received line signal detector lead (V.36)
DTR_ON, DTR_OFF	(--)	Sets the data terminal ready lead (V.28, V.35, V.36)
SQ_ON, SQ_OFF	(--)	Sets the signal quality lead (V.28)
RI_ON, RI_OFF	(--)	Sets the ring indication lead (V.28, V.35) calling indicator (V.36)
DRS_ON, DRS_OFF	(--)	Sets the data signal rate select lead

Table B–2 Setting Leads

Valid Frame and Packet Identifiers					
Frame Type	R*SABM	R*DISC	R*I	R*RRC	R*UA
	R*SABME	R*RR	R*RNR	R*REJ	R*REJC
	R*DM	R*FRMR	R*INVFRM	R*RNRC	
Packet Type	R*DATAP	R*RRP	R*RNRP	R*CALLREQ	R*CALLCON
	R*CLEARREQ	R*CLEARCONF	R*INTREQ	R*INTCONF	R*RESETREQ
	R*RESETCONF	R*RESTARTREQ	R*RESTARTCONF	R*DIAGNOSTIC	R*INVPKT

Table B–3 Valid Frame and Packet Identifiers

Frame and Packet Events		
Command	**Stack Description**	**Description**
?RX	(id #1\...\id #n\n -- flag) where: n=number of identifiers	Detects a frame or packet
?RX_FRAME	(id #1\...\id #n\n -- flag) where: n=number of identifiers	Detects a frame when exclusive layer 2 testing is required
?RX_PACKET	(id #1\...\id #n\n -- flag) where: n=number of identifiers	Detects a packet when exclusive layer 3 testing is required
?FRAME	(-- flag)	Detects any frame
?RX_DATA	(string -- flag)	Detects a specific string in the data field of a data packet
?RECEIVED	(string --)	Detects an exact string
?SEARCH	(string -- flag)	Detects a string anywhere in the frame

Table B–4 Frame and Packet Events

LCN Specific Emulation Setup Commands		
Command	**Stack Description**	**Description**
CHn	(--)	Selects one of the 255 logical channels Example: CH1
=LCN	(LCN number --)	Specifies LCN of currently selected channel
SVC	(--)	Sets currently selected channel as switched virtual circuit
PVC	(--)	Sets currently selected channel as permanent virtual circuit
=CALLED	(string --)	Sets the called address of the selected channel
=CALLING	(string --)	Sets the calling address of the selected channel
=WINDOW	(window size --)	Sets the packet window size
=SIZE	(data field --) 1065	Sets the maximum size of data packets
=CLASS	(class value --)	Sets the throughput class of the selected channel
ECHO_ON, ECHO_OFF	(--)	Sets the echo mode switch for the emulation
MAKE_CUD	(string --) or (0 --) to reset	Sets a call user data field for call request packets
YES_FAC, NO_FAC	(--)	Uses facility negotiation automatically
USER_FAC	(--)	Uses custom, user-defined facility fields
MAKE_FAC	(string --)	Defines custom user-defined facility fields
FAST_SELECT_ON FAST_SELECT_RESTRICTION FAST_SELECT_OFF	(--)	Sets fast select facility option for the emulation
CLEARREQ_EXT CLEARREQ_NOT_EXT	(--)	Uses extended versions of clear request packets
CLEARCONF_EXT CLEARCONF_NOT_EXT	(--)	Uses extended versions of clear confirmation packets
USER_CAFAC, NO_CAFAC	(--)	Selects facility fields usage for call accept packets
MAKE_CAFAC, NO_CAFAC	(string --)	Defines the facility field for call accept packets
ECHO_CAFAC, NO_CAFAC	(--)	Sets the emulation to echo the call request facilities unconditionally in the call accept
YES_CA, NO_CA	(--)	Address field used/not used in call connected packets

Table B–5 LCN Specific Emulation Setup Commands

Sending Frames and Packets		
Command	**Stack Description**	**Description**
SENDF	(string --)	Sends a string as a frame
BUFFER_SENDF	(buffer number --)	Sends a buffer as a frame
SENDP	(string --)	Sends a string as a packet in an I frame
BUFFER_SENDP	(buffer number --)	Sends a buffer as a packet in an I frame
S:INF	(--)	Sends any queued information frame
SENDD	(string --)	Sends a string as the data field of a data packet
BUFFER_SENDD	(buffer number --)	Sends a buffer as a data field of a data packet
DATA, S:DATAP	(--)	Sends a data packet
SABM, S:SABM	(--)	Sends an SABM frame
SABME, S:SABME	(--)	Sends an SABME frame
DISC, S:DISC	(--)	Sends a DISC frame
S:UA	(--)	Sends a UA frame
DM, S:DM	(--)	Sends a DM frame
FRMR, S:FRMR	(--)	Sends an FRMR frame
S:RR	(--)	Sends an RR frame
S:RNR	(--)	Sends an RNR frame
S:REJ	(--)	Sends a REJECT frame
S:RRC	(--)	Sends an RR command frame
S:RNRC	(--)	Sends an RNR command frame
S:REJC	(--)	Sends a REJECT command frame
RESTART, S:RESTARTR	(--)	Sends a restart request packet
S:RESTARTC	(--)	Sends a restart confirm packet
CALL, S:CALLR	(--)	Sends a call request packet
S:CALLC	(--)	Sends a call connect/accept packet
RESET, S:RESETR	(--)	Sends a reset request packet
S:RESETC	(--)	Sends a reset confirmation packet
CLEAR, S:CLEARR	(--)	Sends a clear request packet
S:CLEARC	(--)	Sends a clear confirmation packet
INTERRUPT, S:INTR	(--)	Sends an interrupt request packet
S:INTC	(--)	Sends an interrupt confirmation packet
S:RRP	(--)	Sends an RR packet
S:RNRP	(--)	Sends an RNR packet
CRC_ERROR	(--)	Forces a CRC error on the next frame sent
DO_ABORT	(--)	Forces an abort during the transmission of the next frame

Table B-6 Sending Frames and Packets

Creating Buffers		
Command	Stack Description	Description
FILE->BUFFER	(filename\buffer number --)	Loads a buffer from a file
STRING->BUFFER	(string\buffer number --)	Loads a buffer from a string (maximum 255 bytes)
ALLOT_BUFFER	(size\buffer number -- flag)	Allocates memory for a buffer
FILL_BUFFER	(data address\size\buffer number --)	Moves data into a buffer and overwrites the previous contents
APPEND_TO_BUFFER	(data address\size\buffer number --)	Appends data into a buffer
CLEAR_BUFFER	(buffer number --)	Stores a size of 0 in the buffer
BUFFER	(buffer number -- address)	Returns the address of the first byte of the specified buffer

Table B-7 Creating Buffers

Multilink Commands		
Command	Stack Description	Description
GET_MNS	(-- MNS value)	Obtain MN(S) field from last received MLP frame
GET_V	(-- V bit)	Obtain last received V bit
GET_S	(-- S bit)	Obtain last received S bit
GET_R	(-- R bit)	Obtain last received R bit
GET_C	(-- C bit)	Obtain last received C bit
CAUSE_RECEIVED?	(-- flag)	Determine if last received MLP frame contained a reset cause field
GET_CAUSE	(-- reset cause field)	Obtain reset cause field
LOAD_MNS	(MNS value --)	Sets MN(S) field for next transmitted frame
SMNS	(-- address)	Variable containing MN(S) used when transmitting MLP frames
LOAD_V	(V bit value --)	Sets V bit in transmitted frames
LOAD_S	(S bit value --)	Sets S bit in transmitted frames
LOAD_R	(R bit value --)	Sets R bit in transmitted frames
LOAD_C	(C bit value --)	Sets C bit in transmitted frames
LOAD_CAUSE	(cause field --)	Set contents of reset cause field in transmitted frames
SEND_CAUSE	(flag --)	Specifies if reset cause field is included in transmitted frames
SEND_MLP	(--)	Transmits an MLP I frame constructed with above commands

Table B-8 Multilink Commands

Starting and Examining Timers		
Command	**Stack Description**	**Description**
START_TIMER	(timer#\time--)	Starts an alarm (countdown) timer
STOP_TIMER	(timer# --)	Stops (resets) an alarm timer
START_LAPSE_TIMER	(timer# --)	Starts an elapsed time timer
MINUTES_ELAPSED	(timer# -- minutes)	Examines the minutes elapsed for elapse time timer
SECONDS_ELAPSED	(timer# -- seconds)	Examines the seconds elapsed for elapse time timer
MILLISECONDS_ ELAPSED	(timer# -- milliseconds)	Examines the milliseconds elapsed for elapse time timer

Table B–9 Starting and Examining Timers

Timer Events		
Command	**Stack Description**	**Description**
TIMEOUT	(-- flag)	Detects a timeout of any user timer
?TIMER	(n -- flag)	Detects a timeout of a specific user timer
?WAKEUP	(-- flag)	Detects wakeup timer

Table B–10 Timer Events

Creating User Output		
Command	**Stack Description**	**Description**
T." goes to RAM and Disk too!"	(--) 1 space required after T."	Displays a timestamped comment (trace statement) in the Data Window
TCR	(--)	Inserts a carriage return with the trace statement
T.	(value --)	Displays a decimal value in the Data Window
T.H	(value --)	Displays a hexadecimal value in the Data Window
P." goes to the printer"	(--) 1 space required after the P."	Prints a comment
PCR	(--)	Sends a carriage return to the printer
P.	(value --)	Prints a decimal value
P.H	(value --)	Prints a hexadecimal value

Table B–11 Creating User Output

Program Control Events		
Command	**Stack Description**	**Description**
?KEY	(user function key # --)	Detects a function key
PROMPT" text" actions to be taken using string at address= prompt END_PROMPT	(--)	Prompts the user for keyboard input
?MAIL	(-- flag)	Detects a signal from another ITL program

Table B-12 Program Control Events

C
CODING CONVENTIONS

This section outlines some coding and style conventions recommended by IDACOM. Although you can develop your own style, it is suggested to stay close to these standards to enhance readability.

C.1 Stack Comments

A stack comment is surrounded by parentheses, and shows two stack pictures. The first picture shows any items or 'input parameters' that are consumed by the command; the second picture shows any items or 'output parameters' returned by the command.

Example:
The '=' command has the following stack comment.

(n_1\n_2 -- flag)

In this example, n_1 and n_2 are numbers and the flag is either 0 for a false result, or 1 for a true result. This same example could also be written as follows.

(n_1\n_2 -- 0|1)

The '\' character separates parameters when there is more than one. The parameters are listed from left to right with the leftmost item representing the bottom of the stack and the rightmost item representing the top of the stack.

The '|' character indicates that there is more than one possible output. The above example indicates that either a 0 (false result) or a 1 (true result) is returned on the stack after the '=' operation.

C.2 Stack Comment Abbreviations

Following is a list of commonly used abbreviations. In most cases, the stack comments shown in this manual have been written in full rather than abbreviated.

Symbol	Description
a	Memory address
b	8 bit byte
c	7 bit ASCII character
n	16 bit signed integer
d	32 bit signed integer
u	32 bit unsigned integer
f	Boolean flag (0=false, non–zero=true)
ff	Boolean false flag (zero)
tf	Boolean true flag (non–zero)
s	String (actual address of a character string which is stored in a count prefixed manner)

Table C–1 ITL Symbols

C.3 Program Comments

Program comments appear in source code surrounded by parentheses. These describe the intent or purpose of the definition or line of code.

There must be at least one space on each side of the parentheses.

Example:

```
:  HELLO  ( -- )                   ( Display text Hello in Notice Window )
      " HELLO"                     ( Create string )
      W.NOTICE                     ( Output to Notice Window )
;
```

The program comment should be kept to a minimum and yet contain enough information that another programmer can tell the intent at a glance.

C.4 Test Manager Constructs

Coding conventions for user test scripts should generally follow the style presented throughout this manual.

Indenting nested program structures should be done using the tab key in the editor. The use of many meaningful comments is highly recommended and enhance the continued maintainability of the program.

Example:
(State definition purpose comment)

```
0 STATE{
      EVENT Recognition Commands          ( Comment )
      ACTION{
          Action Commands                 ( Comment )
          IF
              ...                         ( Comment )
              ...                         ( Comment )
          ENDIF
      }ACTION
  }STATE
```

C.5 Spacing and Indentation Guidelines

The following list outlines the general guidelines for spacing and indentations:

- One space between colon and name in colon definitions.
- One space between opening parenthesis and text in comments.
- One space between numbers and words within a definition.
- One space between initial " in strings (i.e. with " string", W." string", T." string", P." string", X" hex characters", etc...)
- One or more spaces at the end of each line unless defining a string which requires additional characters.
- Tab for nested constructs.
- Carriage return after colon definition and stack comment.
- Carriage return after last line of code in colon definition and semi-colon.

See the examples in Appendices C.6 and C.4.

C.6 Colon Definitions

The colon definition should be preceded by a short comment start at the first column of a line. All codes underneath the definition name should be preceded by one tab. Each element within the colon definition should be well defined.

Example:
(Description of command)

```
:  COMMANDNAME                          ( Stack description )
      .....                             ( Comment for first line of code )
      IF
          ....                          ( Comment )
          DOCASE
              CASE  X  { ... }          ( Comment )
              CASE  Y  { ... }          ( Comment )
              CASE  DUP  { ... }        ( Comment )
          ENDCASE
      ELSE
          BEGIN
              .....                     ( Comment )
              .....                     ( Comment )
          UNTIL
      ENDIF
;
```

D
ASCII/EBCDIC/HEX CONVERSION TABLE

HEX	DEC	OCT	ASCII	EBCDIC	HEX	DEC	OCT	ASCII	EBCDIC
00	0	00	NUL	NUL	30	48	60	0	
01	1	01	SOH	SOH	31	49	61	1	
02	2	02	STX	STX	32	50	62	2	SYN
03	3	03	ETX	ETX	33	51	63	3	IR
04	4	04	EOT	PF	34	52	64	4	PP
05	5	05	ENQ	HT	35	53	65	5	TRN
06	6	06	ACK	LC	36	54	66	6	NBS
07	7	07	BEL	DEL	37	55	67	7	EOT
08	8	10	BS	GE	38	56	70	8	SBS
09	9	11	HT	SPS	39	57	71	9	IT
0A	10	12	LF	RPT	3A	58	72	:	RFF
0B	11	13	VT	VT	3B	59	73	;	CU3
0C	12	14	FF	FF	3C	60	74	<	DC4
0D	13	15	CR	CR	3D	61	75	=	NAK
0E	14	16	SO	SO	3E	62	76	>	
0F	15	17	SI	SI	3F	63	77	?	SUB
10	16	20	DLE	DLE	40	64	100	@	SP
11	17	21	DC1	DC1	41	65	101	A	
12	18	22	DC2	DC2	42	66	102	B	
13	19	23	DC3	DC3	43	67	103	C	
14	20	24	DC4	RES	44	68	104	D	
15	21	25	NAK	NL	45	69	105	E	
16	22	26	SYN	BS	46	70	106	F	
17	23	27	ETB	POC	47	71	107	G	
18	24	30	CAN	CAN	48	72	110	H	
19	25	31	EM	EM	49	73	111	I	
1A	26	32	SUB	UBS	4A	74	112	J	cent
1B	27	33	ESC	CUI	4B	75	113	K	.
1C	28	34	FS	IFS	4C	76	114	L	<
1D	29	35	GS	IGS	4D	77	115	M	(
1E	30	36	RS	IRS	4E	78	116	N	+
1F	31	37	US	IUS	4F	79	117	O	|
20	32	40	SP	DS	50	80	120	P	&
21	33	41	!	SOS	51	81	121	Q	
22	34	42	"	FS	52	82	122	R	
23	35	43	#	WUS	53	83	123	S	
24	36	44	$	BYP	54	84	124	T	
25	37	45	%	LF	55	85	125	U	
26	38	46	&	ETB	56	86	126	V	
27	39	47	'	ESC	57	87	127	W	
28	40	50	(	SA	58	88	130	X	
29	41	51	)	SFE	59	89	131	Y	
2A	42	52	*	SM/SW	5A	90	132	Z	!
2B	43	53	+	CSP	5B	91	133	[	$
2C	44	54	,	MFA	5C	92	134	\	
2D	45	55	-	ENQ	5D	93	135	]	)
2E	46	56	.	ACK	5E	94	136	^	;
2F	47	57	/	BEL	5F	95	137	_	¬

HEX	DEC	OCT	ASCII	EBCDIC
60	96	140	`	–
61	97	141	a	/
62	98	142	b	
63	99	143	c	
64	100	144	d	
65	101	145	e	
66	102	146	f	
67	103	147	g	
68	104	150	h	
69	105	151	i	
6A	106	152	j	|
6B	107	153	k	,
6C	108	154	l	%
6D	109	155	m	_
6E	110	156	n	>
6F	111	157	o	?
70	112	160	p	
71	113	161	q	^
72	114	162	r	
73	115	163	s	
74	116	164	t	
75	117	165	u	
76	118	166	v	
77	119	167	w	
78	120	170	x	
79	121	171	y	\
7A	122	172	z	:
7B	123	173	{	#
7C	124	174	|	@
7D	125	175	}	'
7E	126	176	~	=
7F	127	177	DEL	"
80	128	200		
81	129	201		a
82	130	202		b
83	131	203		c
84	132	204		d
85	133	205		e
86	134	206		f
87	135	207		g
88	136	210		h
89	137	211		i
8A	138	212		
8B	139	213		{
8C	140	214		≤
8D	141	215		(
8E	142	216		+
8F	143	217		†
90	144	220		
91	145	221		j
92	146	222		k
93	147	223		l
94	148	224		m
95	149	225		n
96	150	226		o
97	151	227		p
98	152	230		q
99	153	231		r
9A	154	232		
9B	155	233		}
9C	156	234		□
9D	157	235		)
9E	158	236		±
9F	159	237		■
A0	160	240		–
A1	161	241		o
A2	162	242		s
A3	163	243		t
A4	164	244		u
A5	165	245		v
A6	166	246		w
A7	167	247		x
A8	168	250		y
A9	169	251		z
AA	170	252		
AB	171	253		└
AC	172	254		┌
AD	173	255		[
AE	174	256		≥
AF	175	257		•
B0	176	260		0
B1	177	261		1
B2	178	262		2
B3	179	263		3
B4	180	264		4
B5	181	265		5
B6	182	266		6
B7	183	267		7
B8	184	270		8
B9	185	271		9
BA	186	272		
BB	187	273		┘
BC	188	274		┐
BD	189	275		]
BE	190	276		≠
BF	191	277		–

HEX	DEC	OCT	ASCII	EBCDIC
C0	192	300		{
C1	193	301		A
C2	194	302		B
C3	195	303		C
C4	196	304		D
C5	197	305		E
C6	198	306		F
C7	199	307		G
C8	200	310		H
C9	201	311		I
CA	202	312		
CB	203	313		
CC	204	314		
CD	205	315		
CE	206	316		
CF	207	317		
D0	208	320		}
D1	209	321		J
D2	210	322		K
D3	211	323		L
D4	212	324		M
D5	213	325		N
D6	214	326		O
D7	215	327		P
D8	216	330		Q
D9	217	331		R
DA	218	332		
DB	219	333		
DC	220	334		
DD	221	335		
DE	222	336		
DF	223	337		
E0	224	340		\
E1	225	341		
E2	226	342		S
E3	227	343		T
E4	228	344		U
E5	229	345		V
E6	230	346		W
E7	231	347		X
E8	232	350		Y
E9	233	351		Z
EA	234	352		
EB	235	353		
EC	236	354		
ED	237	355		
EE	238	356		
EF	239	357		

HEX	DEC	OCT	ASCII	EBCDIC
F0	240	360		0
F1	241	361		1
F2	242	362		2
F3	243	363		3
F4	244	364		4
F5	245	365		5
F6	246	366		6
F7	247	367		7
F8	248	370		8
F9	249	371		9
FA	250	372		
FB	251	373		
FC	252	374		
FD	253	375		
FE	254	376		
FF	255	377		

E
COMMAND CROSS REFERENCE LIST

This appendix cross references old commands and variables, not appearing in this manual, with new replacement commands. Reference should be made to the previous versions of this manual for description of the old commands. The new commands achieve the same function, however, the input/output parameters may have changed.

Old Command	New Command
C-WANTED-LCN	=CLCN1
CLCN=	=CLCN1
CLCN=SEL	CLCN1=SEL
CTRIG=	C-WCALLED
D-WANTED-LCN	=DLCN1
DATA-STATUS	STATUS_ERR?
DLCN=	=DLCN1
DLCN=SEL	DLCN1=SEL
DTRIG=	D-WCALLED
PLAY-COUNT	BLOCK-COUNT
PLAY-ETIME	END-TIME
PLAY-ID	PORT-ID
PLAY-STIME	START-TIME
R-WANTED-LCN	=RLCN1
REC-STATUS/DATA-STATUS	STATUS_ERR?
RLCN=	=RLCN1
RLCN=SEL	RLCN1=SEL
RTRIG=	R-WCALLED
SET_LONG	LONG-INTERVAL
SET_SHORT	SHORT-INTERVAL
SET_SPEED	INTERFACE-SPEED

INDEX

INDEX [continued]

INDEX [continued]

INDEX [continued]

INDEX [continued]

INDEX [continued]