SDLC/SNA PROGRAMMER'S MANUAL

September 1990
Version 2.0

PREFACE

This manual is intended to provide a programmer's guide to the SNA Monitor and SDLC 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 6000-1203

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

2
MONITOR CONFIGURATION

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

Monitor Configuration Menu	
→ Interface Type	RS232C/V.28
Interface Leads	DISABLED
Bit Rate	UNKNOWN
Clocking	NRZ WITH CLOCK
Frame Sequence Number Modulo	MOD 8

Figure 2–1 Monitor Configuration Menu

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

NOTE
Use WAKEUP_CPU once after all physical changes are made.

→ *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 for protocol testing 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 (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 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*
When NRZI clocking is selected, the interface speed can be selected from preset values on the Interface Port Speed Menu or set to a user-defined speed. When any other clocking mode is selected, the speed is measured, in bits per second, directly from the physical line.

The following commands should only be used when NRZI clocking is selected.

SPEED=50	**SPEED=300**	**SPEED=4800**	**SPEED=38400**
SPEED=75	**SPEED=1200**	**SPEED=7200**	**SPEED=48000**
SPEED=110	**SPEED=1800**	**SPEED=9600** (default)	**SPEED=56000**
SPEED=134.5	**SPEED=2000**	**SPEED=14400**	**SPEED=64000**
SPEED=150	**SPEED=2400**	**SPEED=16000**	**SPEED=72000**
SPEED=200	**SPEED=3600**	**SPEED=19200**	**SPEED=128000**

INTERFACE-SPEED (-- address)
Contains the current bit rate (default value is 9600) and is used by the monitor to calculate throughput measurements.

CK_LEAD_ENABLE (--)
Enables the detection of control leads for the currently selected WAN interface.

→ *Clocking*
IF=NRZ (--)
Selects standard non-return to zero line encoding (default) with DCE provided clocks.

NRZ WITH CLOCK function key

IF=NRZ_EXT (--)
Selects a DTE provided transmit clock on pint 24 of an RS-232C connector.

EXTERNAL TX CLOCK function key

IF=NRZI (--)
Selects the non-return to zero inverted method of encoding with timing information extracted from the data signal.

NRZI function key

IF=NRZI+CLK (--)
Selects the non-return to zero inverted method of encoding with timing information extracted from the provided clock signal.

NRZI WITH CLOCK function key

NOTE
If any of these commands are used when the application is running on a B-Channel, the clocking is forced to IF=NRZ (standard non-return to zero encoding).

→ *Frame Sequence Number Modulo*

L2=MOD8 (--)

Enables modulo 8 method of decoding (default).

MOD 8 function key

L2=MOD128 (--)

Enables modulo 128 method of decoding.

MOD 128 function key

NOTE

The program is automatically placed into modulo 8 when an SNRM is received, and modulo 128 when an SNRME is received.

3
MONITOR ARCHITECTURE

The SNA 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 data displayed, captured, or recorded data.

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.

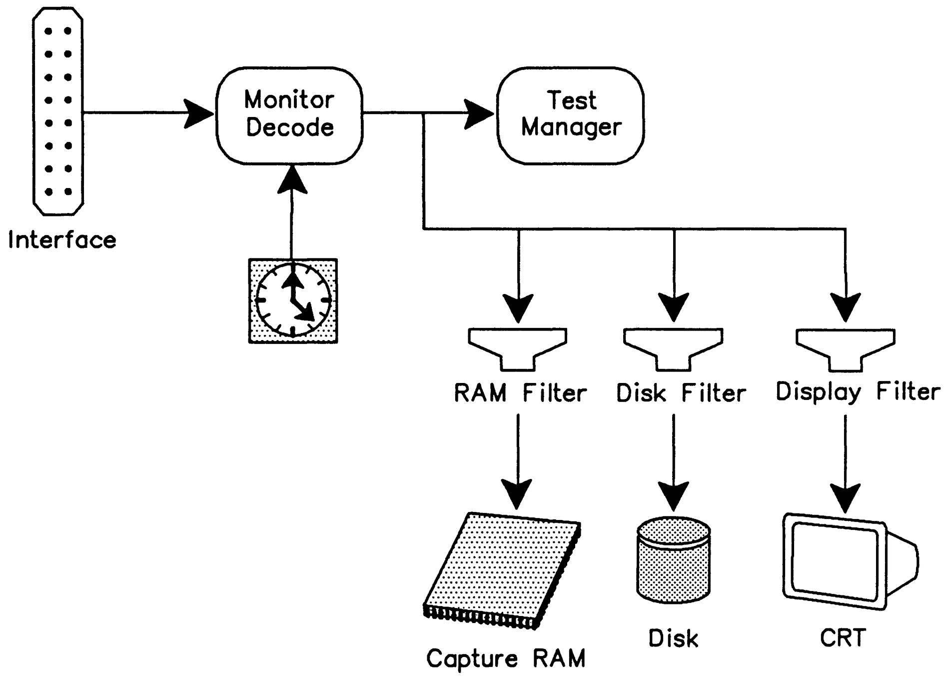


Figure 3–1 SNA Monitor Data Flow Diagram – Live Data

By default, the SNA 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.

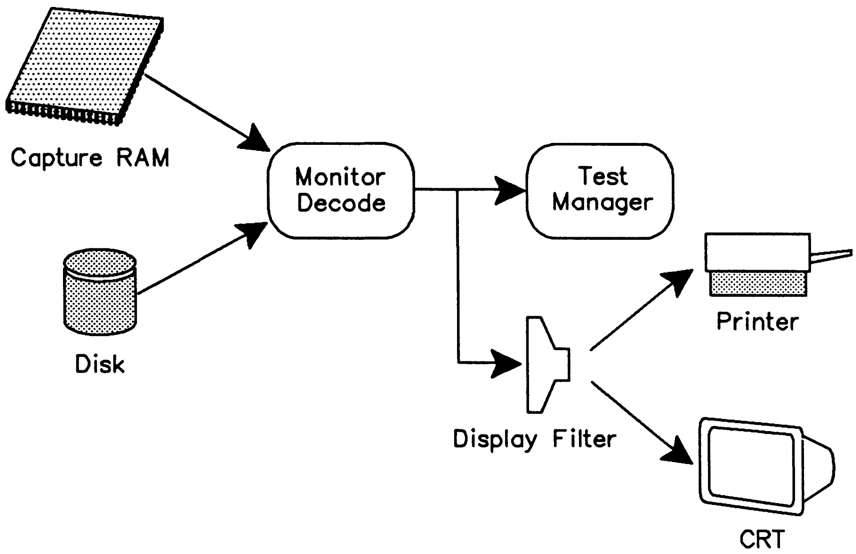


Figure 3–2 SNA Monitor Data Flow Diagram – Offline Processing

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 )
" SNADAT" =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 screen at the beginning of the playback source.

CTRL SHIFT ⇑

BOTTOM (--)
Positions the screen 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.

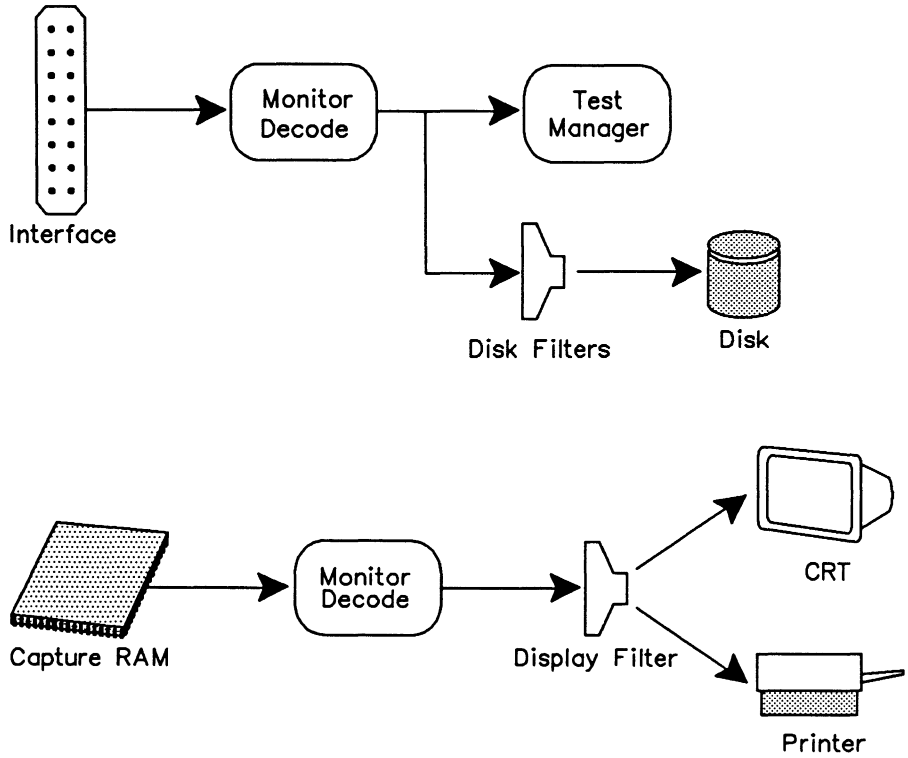


Figure 3–3 SNA Monitor Data Flow Diagram – Freeze Mode

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 SNA Monitor supports both modulo 8 and modulo 128 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.

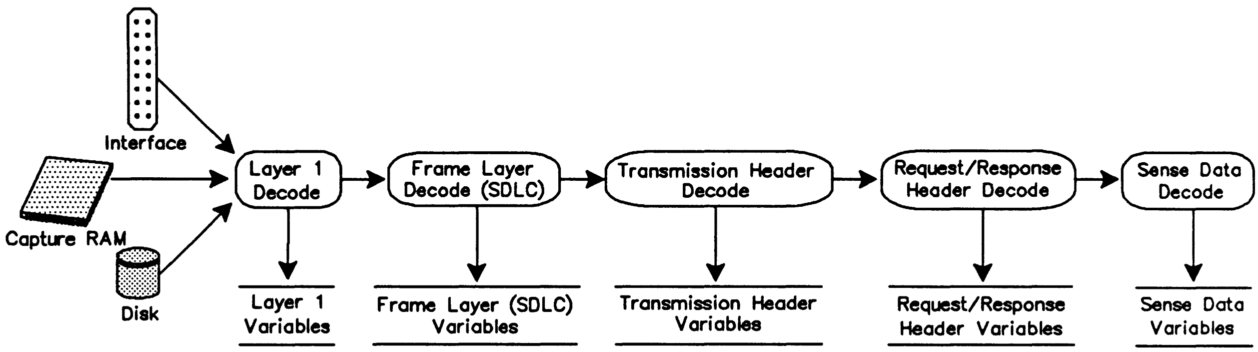


Figure 4–1 SNA Monitor Data Flow Diagram – Decode

4.1 Communication Variables

The following variables are set during the decode process and contain protocol specific information as defined by IBM.

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); for timers, the number of the expired timer.

PORT-ID (-- address)

Contains a 2 byte value identifying received direction for data. The lower byte indicates the TO_DCE or TO_DTE 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 pre-defined 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 command. 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 command. See the START-TIME example.

BLOCK-COUNT (-- address)

Contains the sequential 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. Initially contains a value of zero 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.

Example:
REC-LENGTH @ provides the total number characters in the current frame (BLU).

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 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.

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 XXXX bytes
SHORT_FRM_ERR?	Frame is shorter than XX bytes (including 2 CRC bytes)
CTRL_BYTE_ERR?	Invalid control byte
ADDR_BYTE_ERR?	Invalid address byte
LENGTH_ERR?	Invalid frame length

LEAD-NUMBER (-- address)
Contains the received lead identifier used in the test manager.

Frame (SDLC) Layer

The frame (SDLC) layer decode operation saves information concerning a frame's address, control byte and with I frames, sets up the pointer and length for the transmission header decode.

FRAME-ADDR (-- address)
Contains the PU address (first byte) of the received frame. Valid values are 0 through 255.

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, and second and third bytes for I frames and supervisory frames. See Tables A-1 and A-2 for possible values.

FRAME-MODULO (-- address)
Contains 0 (default) for modulo 8, and 1 for modulo 128. FRAME-MODULO is set to 0 when an SNRM is received, and 1 when an SNRME is received.

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 3 of the control field. In frame modulo 128, the poll/final bit is bit 3 of the control field for unnumbered frames and bit 15 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 0 to 2 of the control field for information and supervisory frames. In frame modulo 128, the N(R) values are bits 8 to 14 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 4 to 6 of the control field for information frames. In frame modulo 128, the N(S) values are bits 0 to 6 of the control field for information frames.

FID–LENGTH (-- address)
Contains the length of the received transmission header and any following bytes. This length does not include the frame layer bytes or the FCS bytes. If the received frame is not an I frame, the length is 0.

FID–POINTER (-- address)
Contains the pointer to byte 0 in the received transmission header.

TH (Transmission Header)

SNA defines six types of TH's (transmission headers): FID0, FID1, FID2, FID3, FID4, and FIDF. The format of each type is determined by the FID (format identifier) which is located in bits 0 to 3 of byte 0 of the TH.

NOTE
Refer to Appendix A.4 for the format of these headers.

Common FID Variables

FID–ID (-- address)
Contains the format identifier field of the transmission header.

ITL Constant	Description
R*FID0	FID Type 0
R*FID1	FID Type 1
R*FID2	FID Type 2
R*FID3	FID Type 3
R*FID4	FID Type 4
R*FIDF	FID Type F
R*INVFID	Invalid FID

MAPPING–FIELD (-- address)
Contains the mapping field located in bits 4 and 5 of byte 0 of FID0, FID1, FID2, and FID3, and bits 4 and 5 of byte 16 of FID4. This field indicates if the message is segmented and which part of the segment it is.

ITL Constant	Description
R*WHOLE–BIU	Whole message
R*FIRST–SEGMENT	First segment of message
R*LAST–SEGMENT	Last segment of message
R*MIDDLE–SEGMENT	Middle or between segment of message

EXP–FLOW–IND (-- address)
Contains the EFI (expedited flow indicator) located in bit 7 of byte 0 of FID0, FID1, FID2, and FID3, and bit 7 of byte 16 of FID4. The EFI informs the LU transmission control layer that the message is to be handled in an expedited fashion.

Value	Description
0	Normal
1	Expedited

DEST–ADD–FIELD (-- address)
Contains the DAF (destination address field). This field, located in bytes 2 and 3 of FID0 and FID1, and byte 2 of FID2, specifies which mode the message is directed. Valid values are 0 through 65535.

ORIG–ADD–FIELD (-- address)
Contains the OAF (origination address field). This field, located in bytes 4 and 5 of FID0 and FID1, and byte 3 of FID2 specifies the mode transmitting the message. Valid values are 0 through 65535.

SEQ–NUM–FIELD (-- address)
Contains the SNF (sequence number field) located in bytes 6 and 7 of FID0 and FID1, bytes 4 and 5 of FID2, and bytes 22 and 23 of FID4. This field is generated by the LU (logical unit) and is used for detection of lost messages. Valid values are 0 through 65535.

DATA–COUNT–FIELD (-- address)
Contains the DCF (data count field) located in bytes 8 and 9 of FID0 and FID1, and bytes 24 and 25 of FID4 and FIDF. The field specifies how many bytes of data comprise the RH (request/response header) and RU (request/response unit).

FID3 Variables

LU–SSCP–IND (-- address)
Contains the LU/SSCP indicator from the LSID (local system ID) located in bit 0 of byte 1 of FID3. The LU/SSCP indicates whether the source of the message was an SSCP (system services control point) or an LU (logical unit).

Hex Value	Description
0	SSCP is the message source
80	LU is the message source

LU–PU–IND (-- address)
Contains the LU/PU indicator from the LSID (local system ID) located in bit 1 of byte 1 of FID3. The LU/PU indicates whether the destination of the message is a PU (physical unit) or an LU (logical unit).

Hex Value	Description
0	Destination is a PU
40	LU is the message source

LOCAL–ADDRESS (-- address)
Contains the local address field from the LSID (local system ID) located in bits 2 to 7 of byte 1 of FID3. This field indicates which LU (logical unit) the message is destined. Valid values are 0 through 63.

FID4 Variables

TG–SWEEP (-- address)
Contains the transmission group sweep indicator located in bit 4 of byte 0 of FID4. Specifies whether PIU's (path information unit) can pass PIU's.

Value	Description
0	PIU's may pass PIU's
8	PIU's may not pass PIU's

ER–VR–SUPP–IND (-- address)
Contains the ER (explicit route) and VR (virtual route) support indicator located in bit 5 of byte 0 of FID4.

Value	Description
0	Each node supports ER and VR
4	Each node does not support ER and VR

VR–PAC–CNT–IND (-- address)
Contains the VR (virtual route) pacing count located in bit 6 of byte 0 of FID4. Pacing is the method by which a receiving component controls the rate of transmission of a sending unit to prevent overrun or congestion.

Value	Description
0	VR pacing count not zero
2	VR pacing count is zero

NTWK–PRIORITY (-- address)
Contains the network priority indicator located in bit 7 of byte 0 of FID4. Indicates whether the flow is less than network priority.

Value	Description
0	Flow is less than network priority
1	Flow is network priority

IERN (-- address)
Contains the value of the IERN (initial explicit route number) located in bits 0 to 3 of byte 2 of FID4. Valid values are 0 through 15.

ERN (-- address)
Contains the value of the ERN (explicit route number) located in bits 4 to 7 of byte 2 of FID4. Valid values are 0 through 15.

VRN (-- address)
Contains the VRN (virtual route number) located in bits 0 to 3 of byte 3 of FID4. Valid values are 0 through 15.

TPF (-- address)
Contains the TPF (transmission priority field) located in bits 6 and 7 of byte 3 of FID4.

Value	Description
0	Low transmission priority
1	Medium transmission priority
2	High transmission priority

VR-CWI (-- address)
Contains the VR CWI (virtual route change window indicator) located in bit 0 of byte 4 of FID4.

Hex Value	Description
0	Increment window size
80	Decrement window size

TG-NON-FIFO-IND (-- address)
Contains the TG FIFO (transmission group FIFO indicator) located in bit 1 of byte 1 of FID4.

Hex Value	Description
0	TG FIFO required
40	TG FIFO not required

VR-SQTI (-- address)
Contains the VR SQTI (virtual route sequence and type indicator) located in bits 2 and 3 of byte 4 of FID4.

Hex Value	Description
0	Non-sequenced, non-supervisor
10	Non-sequenced, supervisor
20	Reserved
30	Singly sequenced

TG-SNF (-- address)
Contains the TGSNF (transmission group sequence number) located in bits 4 to 7 of byte 4 and all of byte 5 of FID4. Valid values are 0 through 4095.

VR-PRQ (-- address)
Contains the VR PRQ (virtual route pacing request) located in bit 0 of byte 6 of FID4.

Hex Value	Description
0	No VR pacing response requested
80	VR pacing response requested

VR-PRS (-- address)
Contains the VR PRS (virtual route pacing response) located in bit 1 of byte 6 of FID4.

Hex Value	Description
0	No VR pacing response
40	VR pacing response sent

VR-CWRI (-- address)
Contains the VR CWRI (virtual route change window reply indicator) located in bit 2 of byte 6 of FID4.

Hex Value	Description
0	Increment window size by 1
20	Decrement window size by 1

VR_RWI (-- address)
Contains the VR RWI (virtual route reset window reply indicator) located in bit 3 of byte 6 of FID4.

Hex Value	Description
0	Do not reset window size
10	Reset window size to minimum

VR-SNF-SEND (-- address)
Contains the virtual route send sequence number located in bits 4 to 7 of byte 6 and all of byte 7 of FID4. Valid values are 0 through 4095.

DSAF (-- address)
Contains the DSAF (destination sub-area address) located in bytes 8 to 11 of FID4.

OSAF (-- address)
Contains the OSAF (origin sub-area address) located in bytes 12 to 15 of FID4.

SNAI (-- address)
Contains the SNA indicator located in bit 4 of byte 16 of FID4.

Hex Value	Description
0	Not SNA
10	SNA

DEF (-- address)
Contains the DEF (destination element field) located in bytes 18 and 19 of FID4.

OEF (-- address)
Contains the OEF (origin element field) located in bytes 20 and 21 of FID4.

FIDF Variables

COMMAND-FORMAT (-- address)
Contains the command format field located in byte 2 of FIDF. Valid values are 0 through 255.

COMMAND-TYPE (-- address)
Contains the command type located in byte 3 of FIDF. Valid values are 0 through 255.

COMMAND–SEQUENCE–NUMBER (-- address)
Contains the command sequence number located in bytes 4 and 5 of FIDF.

RH (Request/Response Header)

RH–POINTER (-- address)
Contains the pointer to the first byte of the request/response header.

RH–LENGTH (-- address)
Contains the length of the request/response header (3 bytes), plus the length of sense data if present (4 bytes), plus the length of the request/response unit and user data.

RH–REQ–RES (-- address)
Contains the request/response indicator located in bit 8 of byte 0 of the request/response header.

ITL Constant	Description
R*RES–HDR	Response
R*REQ–HDR	Request

RH–ID (-- address)
Contains the request/response unit category located in bits 6 and 7 of byte 0 of the request/response header.

ITL Constant	Description
R*FMD	FMD (function management data)
R*NC	NC (network control)
R*DFC	DFC (data flow control)
R*SC	SC (session control)
R*INVRH	Invalid RH

FORMAT-IND (-- address)

Contains the FI (format indicator) located in bit 4 of byte 0 of the request/response header. This indicator defines field formatting for either a LU-LU session or SSCP (network services) session.

Value	LU-LU Session	SSCP Session
0	No FM header follows	Character coded RU (request/response unit)
8	FM header follows	Field formatted RU

SENSE-DATA-IND (--address)

Contains the SDI (sense data included indicator) located in bit 5 of byte 0 of the request/response header.

Value	Description
0	Sense data not included
4	Sense data included (i.e. an error has occurred)

CHAIN-IND (-- address)

Contains the BCI and ECI (begin and end chain indicators) located in bits 6 and 7 of byte 0 of the request header.

Value	Description
0	Middle RU in the chain
1	Last RU in the chain
2	First RU in the chain
3	Only RU in the chain

RESPONSE-TYPE (-- address)

Contains the DR1I and DR2I (definite response indicators 1 and 2), and the ERI (exception response indicator) for a request header. For a response header, it contains the DR1I and DR2I and the RTI (response type indicator). These indicators are located in bits 0, 2, and 3 respectively of byte 1 of the request/response header.

Hex Value	Request Header	Response Header
0	No response requested	Reserved
20	Definite response requested by DR2I	Positive response to DR2I
30	Exception response requested by DR2I	Negative response to DR2I
80	Definite response requested by DR1I	Positive response to DR1I
90	Exception response requested by DR1I	Negative response to DR1I
A0	Definite response requested by DR1I and DR2I	Positive response to DR1I and DR2I
B0	Exception response requested by DR1I and DR2I	Negative response to DR1I and DR2I

QUEUED–RESP–IND (-- address)

Contains the QRI (queued response indicator) located in bit 6 of byte 1 of the request/response header.

Value	Description
0	Response bypasses transmission control queues
2	Enqueues response in transmission control queues

PACING–IND (-- address)

Contains the PI (pacing indicator) located in bit 7 of byte 1 of the request/response header.

Value	Description
0	Not a pacing request
1	Pacing request or response to pacing request

BRACKET–IND (-- address)

Contains the BBI (begin bracket indicator) and EBI (end bracket indicator) fields located in bits 0 and 1 respectively of byte 2 of the request header.

Hex Value	Description
0	Between or within brackets
40	Notification to other half session that the speaker is finished and that the bracket protocol is complete
80	Request by a half session to become speaker of the session
C0	Only chain in the bracket

CHANGE–DIR–IND (-- address)

Contains the CDI (change direction indicator) located in bit 2 of byte 2 of the request header. This indicator is used to avoid contention within HDX–FF (half duplex flip–flop) sessions.

Hex Value	Description
0	No change in direction requested
20	Indicates the sending LU is finished and requests that the receiving LU now become the sending LU (i.e. change direction)

CODE–SEL–IND (-- address)

Contains the CSI (code selection indicator) located in bit 4 of byte 2 of the request header.

Value	Description
0	Code 0
8	Code 1

ENCIPH-DATA-IND (-- address)
Contains the EDI (enciphered data indicator) located in bit 5 of byte 2 of the request header.

Value	Description
0	RU is not enciphered
4	RU is enciphered

PADDED-DATA-IND (-- address)
Contains the PDI (padded data indicator) located in bit 6 of byte 2 of the request header.

Value	Description
0	RU is not padded
2	RU was padded to a length which is an integral multiple of 8 bytes before enciphering

SD (Sense Data)

The decode operation for sense data is called only when the value in the sense data included field of the RH header contains a 1. In this case, the SENSE-DATA-IND variable contains the value of 4.

SD-POINTER (-- address)
Contains the pointer to byte 0 of the sense data.

SD-LENGTH (-- address)
Contains the remaining length of the frame starting at the first byte of the sense data field. This includes 4 bytes of sense data, the request/response unit field, and any user data.

Example:
Obtain the modifier in byte 1 of the sense data field.

```
SD-LENGTH @ 3 >            ( Check to see if sense data field is complete )
IF
     SD-POINTER @ 1+ C@    ( Get the modifier byte )
ENDIF
```

SD–ID (-- address)

Contains the category of sense data located in byte 0 of the sense data field.

ITL Constant	Description
R*USER–SD	User sense data only
R*REQREJ–SD	Request reject
R*REQERR–SD	Request error
R*STERR–SD	State error
R*RHERR–SD	Request header usage error
R*PERR–SD	Path error
R*INVSD	Invalid sense data

NOTE
A value of 0 in this variable indicates that sense data is not present.

Example:
Obtain the sense–code specific information in bytes 2 and 3 of the sense data field.

```
SD-LENGTH @ 3 >              ( Check to see if sense data field is complete )
IF
     SD-POINTER @ 2+ W@      ( Obtain 2 bytes )
ENDIF
```

RU (Request/Response Unit)

RU–POINTER (-- address)

Contains the pointer to byte 0 of the request/response unit.

NOTE
Request/response units contained within an FMD transmission header have a 3 byte network services header.

Example:
Obtain the network services header.

```
RH-ID @ R*FMD =           ( Is the TH an FMD? )
IF
     RU-POINTER @ @       ( Obtain the NS - 32 bits header )
     8 >>#                ( Shift 8 bits to the right to obtain the three byte
                          ( header )
ENDIF
```

WARNING
Request/response units within an FMD transmission header that are character coded, or LU to LU requests, are implementation dependent. The code in the above example should not be used in these cases.

Example:
Obtain the 1 byte request code of the request/response units contained within a DFC, SC, or NC transmission header.

```
RH-ID @ R*DFC =                  ( Is transmission header a DFC? )
IF                               ( Yes )
      RU-POINTER @ C@            ( Get first byte )
ENDIF
```

RU-LENGTH (-- address)
Contains the remaining length of the frame starting at the first byte of the request/response unit. This length includes the length of the request/response unit field and the length of any user data.

5
CAPTURE TO 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. Furthermore, data stored in capture RAM can be transferred to disk.

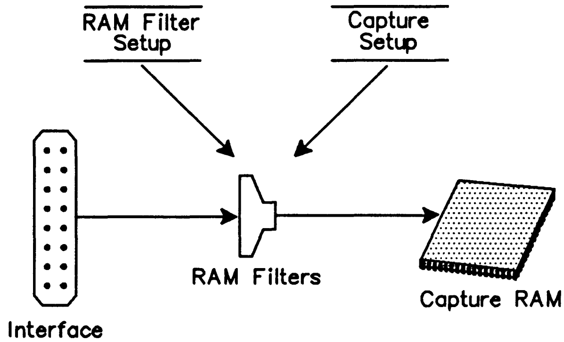


Figure 5–1 SNA Data Flow Diagram – Capture to RAM

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 old data (default) 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.

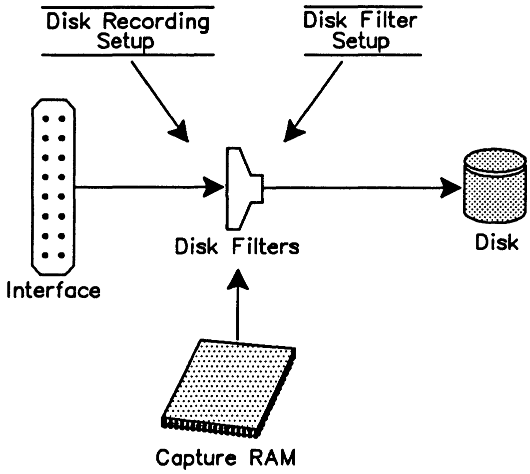


Figure 6–1 SNA Data Flow Diagram – Recording to Disk

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 disk), 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 : Universal Mon    WAN RS232-C
V1.3-1.3 Rev 0                     PT500 - 24     SN# 03-1

Recording End   : Universal Sim    WAN RS232-C
V1.3-1.3 Rev 0                     PT500 - 24     SN# 03-1
```

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 SNA Monitor and SDLC Emulation applications can display data from the line (live data), from capture RAM, or from a disk recording the following four basic display formats:

- Hexadecimal
- Character
- Short
- Complete
- Trace Statements

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

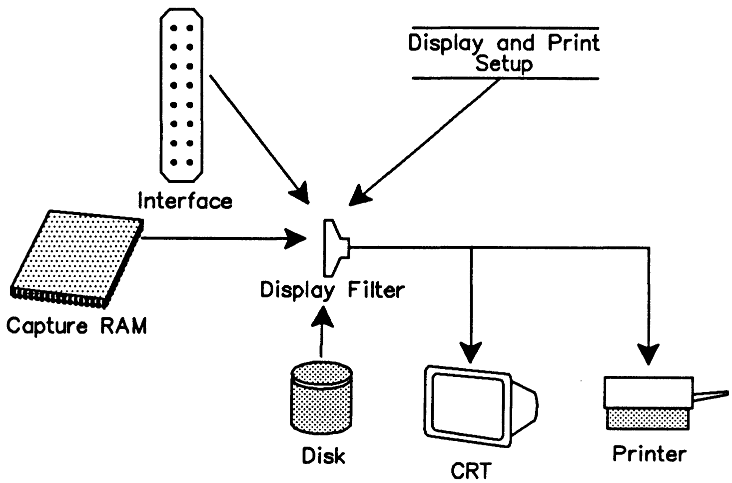


Figure 7–1 SNA Data Flow Diagram – Display and Print

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           EBCDIC
 → Frame Layer             TEXT       Throughput Graph         OFF
   Transmission Header     TEXT         Short Interval (sec)   10
   Request/Resp Header     TEXT         Long  Interval (sec)   600
   Sense Data              TEXT
   Request/Resp Unit       TEXT
   Data Field              CHARACTER
```

Figure 7–2 Display Format Menu

→ Display Format
The default display is short format. *Frame Layer*, *Transmission Header*, *Request/Response Header*, *Sense Data*, *Request/Response Unit*, 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, the PU address, the frame type, the transmission header, request/response header, and request/response unit. 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

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

HEX function key

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

CHARACTER function key

REP_NONE (--)
Displays only trace statements.

Format topic
TRACE function key

→ *Timestamp*
Timestamp reporting is available when *display format* is not in short 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.

ASCII function key

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

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 frame (SDLC) layer data in a detailed report (default).

TEXT function key

FRM_OFF (--)
Frame (SDLC) layer data is not displayed.

OFF function key

FRM_HEX (--)
Displays frame (SDLC) layer data in hex.

HEX function key

FRM_CHAR (--)
Displays frame (SDLC) layer data in the currently selected character set.

CHARACTER function key

→ *Transmission Header*

TH_ON (--)

Displays transmission header information in a detailed report (default).

TEXT function key

TH_OFF (--)

Transmission header information is not displayed.

OFF function key

TH_HEX (--)

Displays transmission header information in hex.

HEX function key

TH_CHAR (--)

Displays transmission header information in the currently selected character set.

CHARACTER function key

→ *Request/Resp Header*

RH_ON (--)

Displays request/response header information in a detailed report (default).

TEXT function key

RH_OFF (--)

Request/response header information is not displayed.

OFF function key

RH_HEX (--)

Displays request/response header information in hex.

HEX function key

RH_CHAR (--)

Displays request/response header information in the currently selected character set.

CHARACTER function key

→ *Sense Data*

SD_ON (--)

Displays sense data information in a detailed report (default).

TEXT function key

SD_OFF (--)

Sense data information is not displayed.

OFF function key

SD_HEX (--)

Displays sense data information in hex.

HEX function key

SD_CHAR (--)

Displays sense data information in the currently selected character set.

CHARACTER function key

→ *Request/Resp Unit*

RU_ON (--)

Displays request/response unit information in a detailed report (default).

TEXT function key

RU_OFF (--)

Request/response unit information is not displayed.

OFF function key

RU_HEX (--)

Displays request/response unit information in hex.

HEX function key

RU_CHAR (--)

Displays request/response unit information in the currently selected character set.

CHARACTER function key

→ *Data Field*

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

CHARACTER function key

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

OFF function key

DATA_HEX (--)
Displays the data field in hex format.

HEX function key

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 SNA 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 (--)
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, and activate or deactivate each of the filters.

```
                      Filter Setup Menu 1

   Filter Type        DISPLAY         Filter Status    ACTIVATED

   Trace Statements   OFF           → Selective PU     193
   Lead Changes       BLOCK           Selective LU     ALL

 Frame Layer:
   SNRM     BLOCK   I      PASS    SIM      BLOCK   CFGR     BLOCK
   DISC     BLOCK   RR     BLOCK   RIM      BLOCK   FRMR     BLOCK
   UA       BLOCK   RNR    BLOCK   BCN      BLOCK   UP       BLOCK
   DM       BLOCK   REJ    BLOCK   XID      BLOCK   Invalid  BLOCK
   RD       BLOCK   UI     BLOCK   TEST     BLOCK
```

Figure 8–1 Filter Setup Menu 1

→ *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, transmission headers, request/response headers, and sense data go to the display.

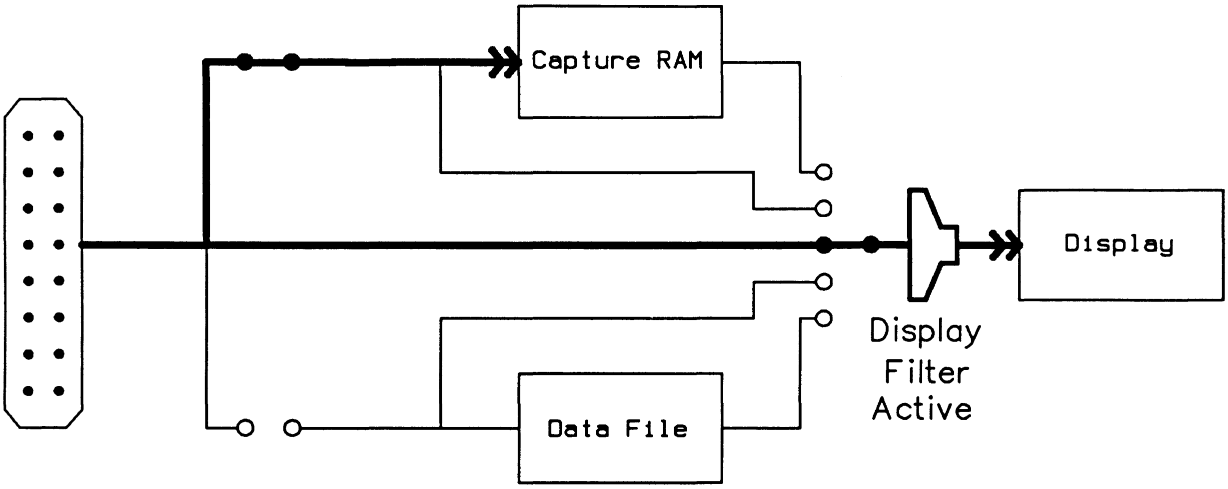


Figure 8–2 Connection Diagram – Display Filters Activated

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 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

→ *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

→ *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 the disk.

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

→ *Selective PU*
The link station address PU (physical unit) is contained in the first byte of all received frames.

If one address has been selected and the display filter is activated, only frames with the specified value in the first byte are displayed unless blocked by another filter.

RFRMA=ALL (––)
Passes all PU's to the display unless blocked by another filter.

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

CFRMA=ALL (––)
Passes all PU's to capture RAM unless blocked by another filter.

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

DFRMA=ALL (--)
Passes all PU's to disk unless blocked by another filter.

→ *Filter Type*
DISK function key
→ *Selective PU*

DO_RFRMA (PU value\1 --)
Specifies the PU address used for display filters. Once display filters are activated, only frames with this address field (first byte of frame) are displayed. Valid PU values are 0 (default) through 255. The '1' must always be included.

Example 1:
Display only traffic to/from physical unit 27.
```
27 1 DO_RFRMA          ( Define selective PU )
REP_FILTER_ON          ( Activate filter )
```

Example 2:
Display only traffic to/from physical unit X'ZZ'.
```
0xZZ 1 DO_RFRMA
```

NOTE
Filters can be set for a hexadecimal station address by prefixing with '0x'.

→ *Filter Type*
DISPLAY function key
→ *Selective PU*
ONE function key

DO_CFRMA (PU value\1 --)
Specifies the PU address used for RAM filters. Once RAM filters are activated, only frames with this address field (first byte of frame) are captured. Valid PU values are 0 (default) through 255. The '1' must always be included.

Example:
Capture only traffic to/from physical unit 54.
```
54 1 DO_CFRMA          ( Define selective PU )
RAM_FILTER_ON          ( Activate filter )
```

→ *Filter Type*
RAM function key
→ *Selective PU*
ONE function key

DO_DFRMA (PU value\1 --)
Specifies the PU address used for disk filters. Once disk filters are activated, only frames with this value in the address field (first byte of frame) are recorded. Valid PU values are 0 (default) through 255. The '1' must always be included.

→ *Filter Type*
DISK function key
→ *Selective PU*
ONE function key

Example:
Record only traffic to/from physical unit 100 to disk.

```
100 1 DO_DFRMA          ( Define selective PU )
DISK_FILTER_ON          ( Activate filter )
```

→ *Selective LU*
The LU (logical unit) is either the destination or origination address field in an FID0, FID1, or FID2 transmission header. This is not applicable for FID3, FID4 or FIDF.

NOTE
The selective LU and selective PU work in an 'AND' condition (i.e. both the selective LU and PU must match in order to pass filters).

RTHAF=ALL
Passes all DAF and OAF (destination and origination) addresses to the display.

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

CTHAF=ALL
Passes all DAF and OAF (destination and origination) addresses to capture RAM.

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

DTHAF=ALL
Passes all DAF and OAF (destination and origination) addresses to disk.

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

DO_RTHAF (RU value/1 --)

Specifies the selective LU address field for display filters. Once display filters are activated, only FID's (FID0, FID1, or FID2) with this value in either the destination or origination address field are displayed. Valid values are 0 through 65535 for FID0 and FID1, and 0 through 255 for FID2. The '1' must always be included.

Example:
Set a display filter for a selective LU to a value of 75.

```
75 1 DO_RTHAF          ( Define selective LU address )
REP_FILTER_ON          ( Activate display filters )
```

→ *Filter Type*
DISPLAY function key
→ *Selective LU*
ONE function key

DO_CTHAF (LU value\1 --)

Specifies the selective LU address field for RAM filters. Once RAM filters are activated, only FID's (FID0, FID1, or FID2) with this value in either the destination or origination address field are captured to RAM. Valid values are 0 through 65535 for FID0 and FID1, and 0 through 255 for FID2. The '1' must always be included.

Example:
Set a RAM filter for a selective LU to 100.

```
100 1 DO_CTHAF           ( Define selective LU address )
RAM_FILTER_ON            ( Activate RAM filter )
```

→ *Filter Type*
RAM function key
→ *Selective LU*
ONE function key

DO_DTHAF (LU value\1 --)

Specifies the selective LU address field for disk filters. Once disk filters are activated, only FID's (FID0, FID1, or FID2) with this value in either the destination or origination address field are recorded to disk. Valid values are 0 through 65535 for FID0 and FID1, and 0 through 255 for FID2. The '1' must always be included.

Example:
Set a disk filter for a selective LU to 2.

```
2 1 DO_DTHAF             ( Define selective LU address )
DISK_FILTER_ON           ( Activate RAM filter )
```

→ *Filter Type*
DISK function key
→ *Selective LU*
ONE function key

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

NOTE
If the I frame has been set to a BLOCK status, a filter at higher layers is inappropriate. This is indicated by the dashes shown beside the individual items.

NOTE
Commands for display filters and SNRM frames are described here as an example. For a complete list of commands, see Table 8–1.

RFRM+SNRM
Passes SNRM frames to the display.

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

RFRM–SNRM
Blocks SNRM frames from the display.

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

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

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

RFRM=NONE
Blocks all frames and packets from the display.

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

Description		Display	RAM	Disk
All	Pass	RFRM=ALL	CFRM=ALL	DFRM=ALL
	Block	RFRM=NONE	CFRM=NONE	DFRM=NONE
Set Normal Response Mode	Pass	RFRM+SNRM	CFRM+SNRM	DFRM+SNRM
	Block	RFRM-SNRM	CFRM-SNRM	DFRM-SNRM
Disconnect	Pass	RFRM+DISC	CFRM+DISC	DFRM+DISC
	Block	RFRM-DISC	CFRM-DISC	DFRM-DISC
Unnumbered Acknowledgment	Pass	RFRM+UA	CFRM+UA	DFRM+UA
	Block	RFRM-UA	CFRM-UA	DFRM-UA
Disconnect Mode	Pass	RFRM+DM	CFRM+DM	DFRM+DM
	Block	RFRM-DM	CFRM-DM	DFRM-DM
Frame Reject	Pass	RFRM+FRMR	CFRM+FRMR	DFRM+FRMR
	Block	RFRM-FRMR	CFRM-FRMR	DFRM-FRMR
Receiver Ready	Pass	RFRM+RR	CFRM+RR	DFRM+RR
	Block	RFRM-RR	CFRM-RR	DFRM-RR
Receiver Not Ready	Pass	RFRM+RNR	CFRM+RNR	DFRM+RNR
	Block	RFRM-RNR	CFRM-RNR	DFRM-RNR
Reject	Pass	RFRM+REJ	CFRM+REJ	DFRM+REJ
	Block	RFRM-REJ	CFRM-REJ	DFRM-REJ
Set Initialization Mode	Pass	RFRM+SIM	CFRM+SIM	DFRM+SIM
	Block	RFRM-SIM	CFRM-SIM	DFRM-SIM
Request Initialization Mode	Pass	RFRM+RIM	CFRM+RIM	DFRM+RIM
	Block	RFRM-RIM	CFRM-RIM	DFRM-RIM
Beacon	Pass	RFRM+BCN	CFRM+BCN	DFRM+BCN
	Block	RFRM-BCN	CFRM-BCN	DFRM-BCN
Configure	Pass	RFRM+CFGR	CFRM+CFGR	DFRM+CFGR
	Block	RFRM-CFGR	CFRM-CFGR	DFRM-CFGR
Request Disconnect	Pass	RFRM+RD	CFRM+RD	DFRM+RD
	Block	RFRM-RD	CFRM-RD	DFRM-RD
Exchange Station Identification	Pass	RFRM+XID	CFRM+XID	DFRM+XID
	Block	RFRM-XID	CFRM-XID	DFRM-XID
Unnumbered Poll	Pass	RFRM+UP	CFRM+UP	DFRM+UP
	Block	RFRM-UP	CFRM-UP	DFRM-UP
Test	Pass	RFRM+TEST	CFRM+TEST	DFRM+TEST
	Block	RFRM-TEST	CFRM-TEST	DFRM-TEST
Unnumbered Information	Pass	RFRM+UI	CFRM+UI	DFRM+UI
	Block	RFRM-UI	CFRM-UI	DFRM-UI
Information	Pass	RFRM+I	CFRM+I	DFRM+I
	Block	RFRM-I	CFRM-I	DFRM-I
Invalid	Pass	RFRM+INV	CFRM+INV	DFRM+INV
	Block	RFRM-INV	CFRM-INV	DFRM-INV

Table 8–1 Frame Layer Filter Commands

```
                         Filter Setup Menu 2

  Filter Type   DISPLAY                  Filter Status   DEACTIVATED

Transmission Header:
→ FID 0     BLOCK    FID 1        BLOCK    FID 2     PASS     FID 3    BLOCK
  FID 4     BLOCK    FID F        BLOCK

Request/Response Header:
  FMD       PASS     NC           BLOCK    DFC       BLOCK    SC       BLOCK
  Invalid   BLOCK

Sense Data:
  REQ REJ   PASS     USER         PASS     REQ ERR   PASS     ST ERR   PASS
  RH USAGE  PASS     PATH ERROR   PASS     Invalid   PASS
```

Figure 8–3 Filter Setup Menu 2

Transmission Header:
If display filters are activated, any FID with a PASS status is displayed unless the selective PU address or the selective LU has been set to a specific value. See the descriptions under *Selective PU* and *Selective LU.*

NOTE
Commands for display filters and FID0 transmission headers are described here as an example. For a complete list of commands, see Table 8–2.

RFID+0
Passes FID0 transmission headers to the display.

→ *Filter Type*
DISPLAY function key
→ *FID0*
PASS function key

RFID–0
Blocks FID0 transmission headers from the display.

→ *Filter Type*
DISPLAY function key
→ *FID0*
BLOCK function key

RFID=ALL

Passes all transmission headers to the display.

→ *Filter Type*
DISPLAY function key
→ *FIDO*
ALL FID function key

RFID=NONE

Blocks all transmission headers from the display.

→ *Filter Type*
DISPLAY function key
→ *FIDO*
NO FID function key

Description		Display	RAM	Disk
All	Pass	RFID=ALL	CFID=ALL	DFID=ALL
	Block	RFID=NONE	CFID=NONE	DFID=NONE
TH FID type = 0	Pass	RFID+0	CFID+0	DFID+0
	Block	RFID–0	CFID–0	DFID–0
TH FID type = 1	Pass	RFID+1	CFID+1	DFID+1
	Block	RFID–1	CFID–1	DFID–1
TH FID type = 2	Pass	RFID+2	CFID+2	DFID+2
	Block	RFID–2	CFID–2	DFID–2
TH FID type = 3	Pass	RFID+3	CFID+3	DFID+3
	Block	RFID–3	CFID–3	DFID–3
TH FID type = 4	Pass	RFID+4	CFID+4	DFID+4
	Block	RFID–4	CFID–4	DFID–4
TH FID type = F	Pass	RFID+F	CFID+F	DFID+F
	Block	RFID–F	CFID–F	DFID–F

Table 8–2 Transmission Header Filter Commands

Request/Response Header:

If display filters are activated, any request/response header with a PASS status is displayed unless the selective PU address or the selective LU has been set to a specific value. See the description under the *Selective Address PU* and *Selective LU.*

NOTE

Commands for display filters and FMD request/response headers are described here as an example. For a complete list of commands, see Table 8–3.

RRH+FMD

Passes FMD request/response headers to the display.

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

RRH−FMD

Blocks FMD request/response headers from the display.

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

RRH=ALL

Passes all request/response headers to the display.

→ *Filter Type*
DISPLAY function key
→ *FMD*
ALL REQ/RES function key

RRH=NONE

Blocks all request/response headers from the display.

→ *Filter Type*
DISPLAY function key
→ *FMD*
NO REQ/RES function key

Description		Display	RAM	Disk
All	Pass	RRH=ALL	CRH=ALL	DRH=ALL
	Block	RRH=NONE	CRH=NONE	DRH=NONE
Function Management Data RH	Pass	RRH+FMD	CRH+FMD	DRH+FMD
	Block	RRH−FMD	CRH−FMD	DRH−FMD
Network Control RH	Pass	RRH+NC	CRH+NC	DRH+NC
	Block	RRH−NC	CRH−NC	DRH−NC
Data Flow Control RH	Pass	RRH+DFC	CRH+DFC	DRH+DFC
	Block	RRH−DFC	CRH−DFC	DRH−DFC
Session Control RH	Pass	RRH+SC	CRH+SC	DRH+SC
	Block	RRH−SC	CRH−SC	DRH−SC
Invalid RH	Pass	RRH+INV	CRH+INV	DRH+INV
	Block	RRH−INV	CRH−INV	DRH−INV

Table 8−3 Request/Response Header Filter Commands

Sense Data:
If display filters are activated, any sense data with a PASS status is displayed unless the selective PU or selective LU has been set to a specific value. See the description under the *Selective Address* and *Selective LU*.

NOTE
Commands for display filters and path error sense data are described here as an example. For a complete list of commands, see Table 8-4.

RSD+PER
Passes path error sense data to the display.

→ *Filter Type*
DISPLAY function key
→ *PATH ERROR*
PASS function key

RSD−PER
Blocks path error sense data from the display.

→ *Filter Type*
DISPLAY function key
→ *PATH ERROR*
BLOCK function key

RSD=ALL
Passes all sense data to the display.

→ *Filter Type*
DISPLAY function key
→ *PATH ERROR*
ALL SD function key

RSD=NONE
Blocks all sense data from the display.

→ *Filter Type*
DISPLAY function key
→ *PATH ERROR*
NO SD function key

NOTE
A frame containing both request/response header and sense data can only be blocked if both are blocked.

Description		Display	RAM	Disk
All	Pass	RSD=ALL	CSD=ALL	DSD=ALL
	Block	RSD=NONE	CSD=NONE	DSD=NONE
Request Reject	Pass	RSD+REQRJ	CSD+REQRJ	DSD+REQRJ
	Block	RSD−REQRJ	CSD−REQRJ	DSD−REQRJ
User	Pass	RSD+USER	CSD+USER	DSD+USER
	Block	RSD−USER	CSD−USER	DSD−USER
Request Error	Pass	RSD+REQER	CSD+REQER	DSD+REQER
	Block	RSD−REQER	CSD−REQER	DSD−REQER
State Error	Pass	RSD+STER	CSD+STER	DSD+STER
	Block	RSD−STER	CSD−STER	DSD−STER
RH Usage Error	Pass	RSD+RHER	CSD+RHER	DSD+RHER
	Block	RSD−RHER	CSD−RHER	DSD−RHER
Path Error	Pass	RSD+PER	CSD+PER	DSD+PER
	Block	RSD−PER	CSD−PER	DSD−PER
Invalid	Pass	RSD+INV	CSD+INV	DSD+INV
	Block	RSD−INV	CSD−INV	DSD−INV

Table 8−4 Sense Data Filter Commands

9
EMULATION CONFIGURATION

This section displays the Emulation Configuration Menu and describes commands corresponding to each item.

```
                         Emulation Configuration Menu
  → Emulation Mode   SECONDARY

  Physical Layer:
    Emulation Interface  TO DCE          Bit Rate            UNKNOWN
    Interface Type       RS232C/V.28     Clocking            NRZ WITH CLOCK
    Interface Leads      DISABLED        Interframe Fill     FLAG

  Frame Layer:
    Emulation            AUTOMATIC       Multipoint          ---
    Max Frame Size       4096            Two Way Mode        ALTERNATE
    Secondary Stations   32              Poll Timer (Sec)    1.0
```

Figure 9–1 Emulation Configuration Menu

STARTUP (--)
Initializes the SDLC Emulation and configures the physical layer.

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

9.1 Emulation Mode

PRIMARY (--)
Emulates a primary SDLC device and starts the primary polling timer. Selects the primary state machine and forces the emulation into the NO_LINK state (see Table 12–1).

PRIMARY function key

SECONDARY (--)
Emulates a secondary SDLC device. Selects the secondary state machine (default) and forces the emulation into the NDM state (see Table 12–2).

SECONDARY function key

9.2 Physical Layer

The emulation is active at all times. i.e. automatic responses to protocol events are generated even during parameter setup.

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

NOTE
Refer to Table 9-1 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 for protocol testing and electrically isolates the other connectors on the port.

RS422/V.11 function key

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

V.35 function key

IF=V36 (--)
Selects the V.36/RS-449 connector for protocol testing 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 (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
```

ENABLE function key

Example 2:
Disable all leads for the current interface.

```
ALL_LEADS DISABLE_LEAD
```

DISABLE 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.

Clocking	TO DTE	TO DCE
NRZ WITH CLOCK	Select	Measure
EXTERNAL TX CLOCK	Measure	Select
NRZI	Select	Select
NRZI WITH CLOCK	Select	Measure

Table 9–1 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.

The following commands can be used whenever the bit rate can be selected.

SPEED=50
SPEED=75
SPEED=110
SPEED=134.5
SPEED=150
SPEED=200
SPEED=300
SPEED=1200
SPEED=1800
SPEED=2000
SPEED=2400
SPEED=3600
SPEED=4800
SPEED=7200
SPEED=9600 (default)
SPEED=14400
SPEED=16000
SPEED=19200
SPEED=38400
SPEED=48000
SPEED=56000
SPEED=64000
SPEED=72000
SPEED=128000

Example:
Set the bit rate of the emulation to 64000.

```
SPEED=64000
```

INTERFACE–SPEED (--address)
Contains the current bit rate (default is 9600).

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.

→ *Clocking*
IF=NRZ (--)
Expects standard non–return to zero line encoding (default) with DCE provided clocks.

NRZ WITH CLOCK function key

IF=NRZ_EXT (--)
Selects a DTE provided transmit clock on pint 24 of an RS–232C connector.

EXTERNAL TX CLOCK function key

IF=NRZI (--)
Selects the non–return to zero inverted method of encoding with timing information extracted from the data signal.

NRZI function key

IF=NRZI+CLK (--)
Selects the non–return to zero inverted method of encoding with timing information extracted from the provided clock signal.

NRZI WITH CLOCK function key

NOTE
If any of these commands are used when the application is running on a B–Channel, the clocking is forced to IF=NRZ.

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

IDLE_MARK (--)
Sends a mark condition during the interframe idle times.

MARK function key

IDLE_FLAG (--)
Sends flags (hex 7E) during the interframe idle times (default).

FLAG function key

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

```
IF=V35                         ( Use V.35 test connector )
PRIMARY                        ( Emulate a primary SDLC device )
TO_DCE_IF                      ( Set for full DTE simulation )
56000 INTERFACE-SPEED !        ( Set baud rate )
IF=NRZ                         ( Clocking provided by DCE )
STARTUP                        ( Configure software )
```

9.3 Frame Layer

IDACOM's SDLC Emulation implements an automatic layer 2 state machine.

→ *Emulation*
EMUL_OFF (--)
Disables the automatic emulation; the SDLC state machine does not transmit any frames without manual or test program intervention.

MANUAL function key

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

AUTOMATIC function key

→ *Max Frame Size*
MAX-FRAME-LENGTH (-- address)
Contains the maximum frame size. Valid values are 1 through 4096 (default).

Example:
Set the maximum frame size to 2000.
```
2000  MAX-FRAME-LENGTH !
```

→ *Secondary Stations*
DO_#LINKS (number of secondaries\1--)
Specifies the number of active secondary stations that can be emulated simultaneously. Valid values are 1 through 32 (default). The '1' must always be included.

Example:
Set the number of secondary stations to 20.
```
20 1 DO_#LINKS
```

A safer method of setting the number of secondary stations to 20 is:
```
20 DUP 1 32 BETWEEN?        ( Check to see if value is between 1 and 32 )
IF
      1 DO_#LINKS
ELSE
      DROP
ENDIF
```

Modify Number of Active Secondary Stations function key

→ *Multipoint*
PT_TO_PT (--)
Selects point-to-point configuration (default). In this mode, only one secondary can transmit a batch of messages at a time.

OFF function key

MULTI_PT (--)
Selects multipoint configuration. All the links can have active transmissions at the same time.

ON function key

→ *Two Way Mode*
TWA (--)
Uses two way alternate mode (default) for all transmissions (also referred to as half duplex mode).

ALTERNATE function key

TWS (--)
Uses two way simultaneous mode for all transmissions (also referred to as full duplex mode).

SIMULTANEOUS function key

→ *Poll Timer (Sec)*
DO_PTIME (value\1 --)
Specifies the timeout, in tenths of seconds, of the poll timer for primary emulation (default is 1 second).

If the value is 0, the poll timer is forced to one-tenth of a second and is then restarted. The '1' must always be included.

Example:
Set the poll timer to 2 seconds.
```
20 1 DO_PTIME
```

Modify Primary Poll Timer function key

9.4 Station Setup

Individual data links can be configured via the Station Setup Menu. The SDLC Emulation supports 32 stations.

Station Setup Menu

Secondary Station	Address	Window	Timeout(Sec)	Retries
→ Secondary 0	1	4	3.0	20
Secondary 1	2	4	3.0	20
Secondary 2	3	4	3.0	20
Secondary 3	4	4	3.0	20
Secondary 4	5	4	3.0	20
Secondary 5	6	4	3.0	20
Secondary 6	7	4	3.0	20
Secondary 7	8	4	3.0	20
Secondary 8	9	4	3.0	20
Secondary 9	A	4	3.0	20

Figure 9–2 Station Setup Menu

The number of supported secondary stations is specified with the DO_#LINKS command.

NOTE
The commands for selecting a station and specifying the station, address and window size can be used in both primary and secondary emulations. The commands for specifying the timeout and retries apply only to a primary emulation.

DO_LINK# (station number\1--)
Specifies the seccondary station for configuration. Valid values are 0 through 31. If the specified station number is greater than the actual number supported, the emulation selects the last station supported. The '1' must always be included.

Example:
Select secondary station 30.

```
30 1 DO_LINK#
```

SET_LINK_ID (station#\station address --)

Specifies the address of the secondary station. Valid values are 1 through 254.

If the specified station number is greater than the actual number supported, the emulation selects the last station supported.

Example:
Set the address for secondary station 20 to hex 10.

```
20 0X10 SET_LINK_ID
```

Address function key

DO_SETW (window size\1--)

Specifies the window size (number of frames sent before an acknowledgment is received) of the currently selected secondary station. Valid values are 1 through 7. The '1' must always be included.

Example:
Set the window size on station 0 to 5.

```
0 1 DO_LINK#          ( Select station 0 )
5 1 DO_SETW           ( Set the window size to 5 )
```

Window function key

DO_SETTO (tenths of seconds/1--)

Specifies the timeout, in tenths of seconds, the primary waits before retrying a secondary after receiving no response (default is 3 seconds). The '1' must always be included.

Example:
Set the time interval between retries for station number 6 to 5 seconds.

```
6 1 DO_LINK#            ( Select station 6 )
50 1 DO_SETTO           ( Set the time interval to 5 seconds )
```

Timeout function key

DO_SETRETRY (#of retries\1--)

Specifies the number of times the primary polls a selected secondary after receiving no response (default is 20 retries). The '1' must always be included.

Example:
Set the number of retries to 10 for station 4.

```
4 1 DO_LINK#            ( Select station 4 )
10 1 DO_SETRETRY        ( Set number of retries to 10 )
```

Retries function key

10
EMULATION ARCHITECTURE

This section describes the structure of the SDLC Emulation. The IDACOM SDLC Emulation program is a combination of the complete SNA monitor application package together with an SDLC state machine. All commands available in the SNA Monitor are also available in the SDLC 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 SDLC 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 SDLC state machine.

10.1 Live Data

The SNA Monitor decodes any received/transmitted frames and displays them for user interpretation while the SDLC 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.

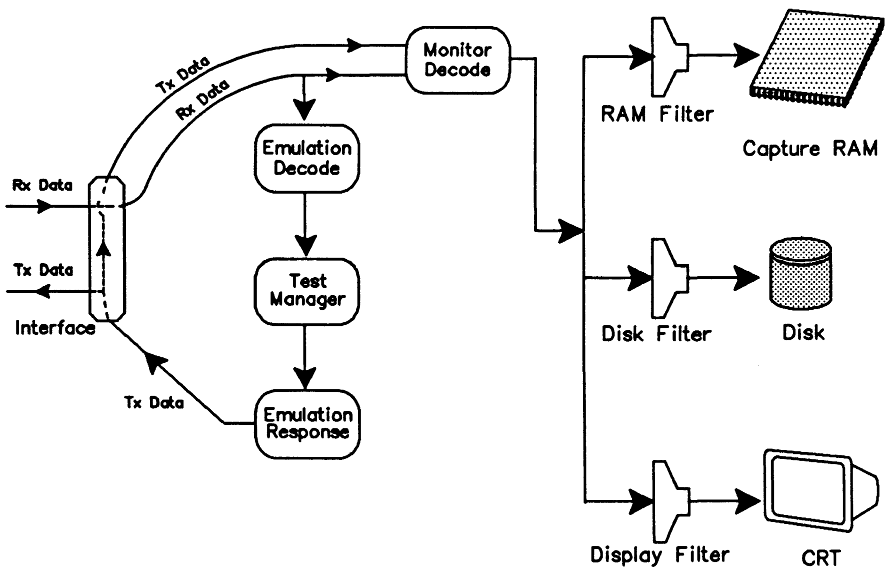


Figure 10-1 SDLC Emulation Data Flow Diagram - Live Data

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

Display topic
Live Data function key

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

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

Emulation topic
Run Emulation function key (highlighted)

EMUL_OFF (--)
Disables the automatic emulation; the SDLC 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.

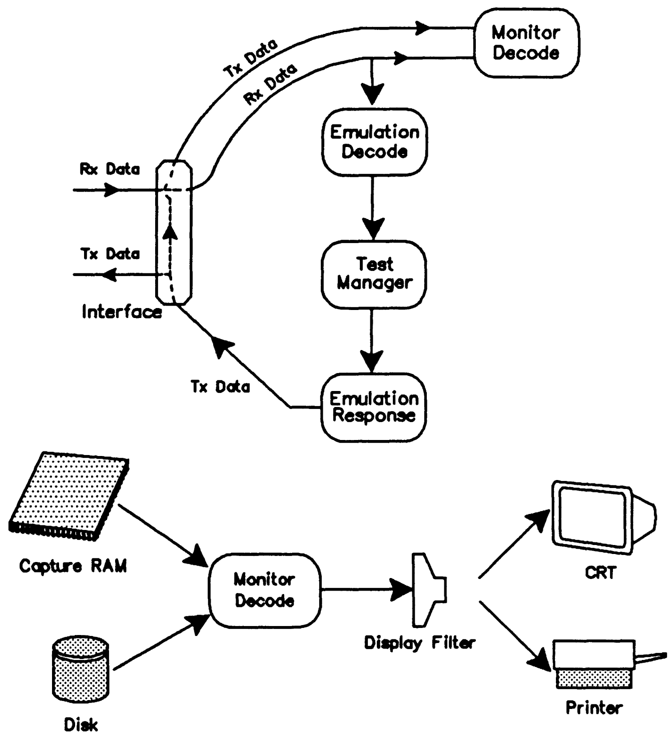


Figure 10–2 SDLC Emulation Data Flow Diagram – Offline Processing

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 the capture RAM or disk file and decoded and displayed or printed. RAM capture 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.

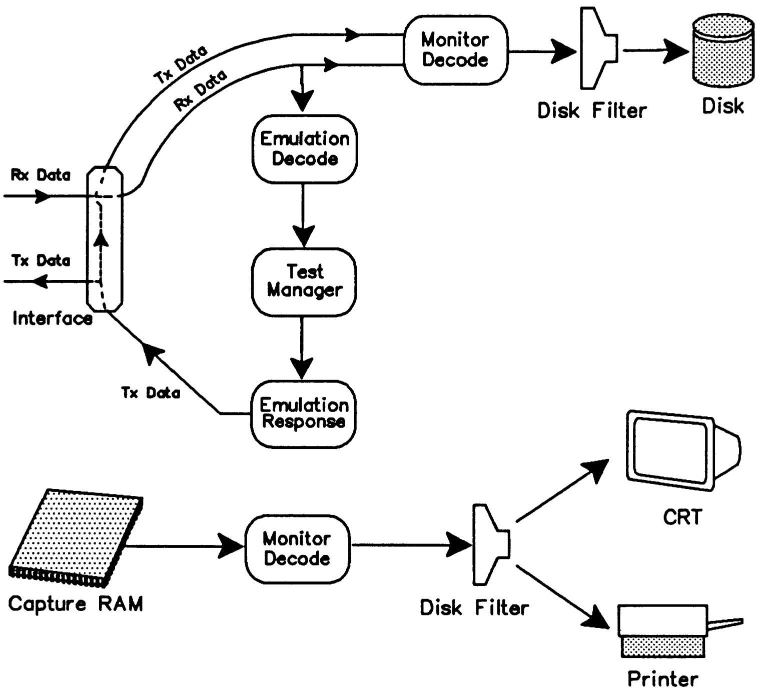


Figure 10–3 SDLC Emulation Data Flow Diagram – Freeze Mode

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 SDLC Emulation is capable of modulo 8 decoding and response only. The standards followed have been defined by IBM.

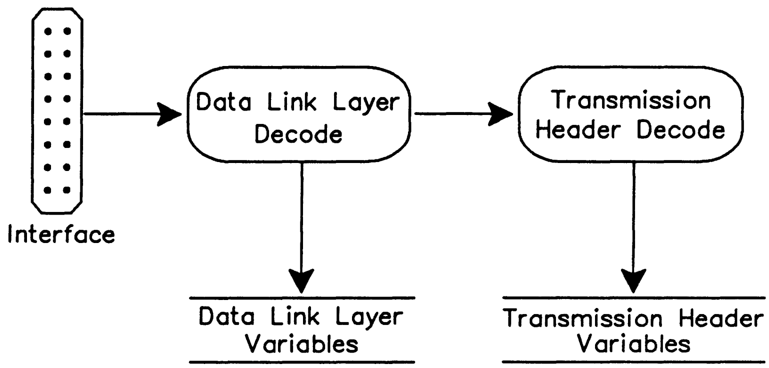


Figure 11-1 SDLC Emulation Data Flow Diagram – Decode

11.1 Communications 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 IBM's publication GA27-3093-2, IBM Synchronous Data Link Control, General Information. 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.

Data Link Receive

ID (-- address)
Contains the address byte of the last received frame. The contents of this variable must be obtained by using C@. Valid values are 0 through 255.

CONTROL (-- address)
Contains the control byte of the last received frame. The contents of this variable must be obtained by using C@. Valid values are shown in Tables A–1 and A–2.

L2–FRAME–POINTER (-- address)
Contains the pointer to the received frame.

Example:
Obtain the control byte of the last received frame (there are three methods).

```
L2-FRAME-POINTER @ 1+ C@      ( From the frame pointer )
```
or,
```
CONTROL C@                    ( From the emulation decode )
```
or,
```
M-CONTROL @                   ( From the monitor decode )
```

NR (--)
Contains the N(R) (received sequence count) of the last received supervisory or information frame. Valid values are 0 through 7. The SDLC Emulation performs checking procedures to determine the appropriate action.

NS (--)
Contains the N(S) (send sequence count) of the last received Information frame. Valid values are 0 through 7. The SDLC Emulation performs checking procedures to determine the appropriate action.

Data Link Transmit

LINK# (--address)
Contains the currently selected link for frame transmission. Valid values are 0 through 31.

#SECS (--address)
Contains the number of secondary stations supported by the emulation. Valid values are 1 through 32.

ID–OUT (-- address)
Contains the current PU address. This address is used in the 'S:' commands for transmitting frames (see Section 12). Valid values are 0 through 255.

BYTE–COUNT (-- address)
Contains the length of the frame to be transmitted.

P/F_BIT (-- address)
Contains the P/F bit used when transmitting frames with the 'S:' commands (see Section 12). Valid values are 0 and 1 (default).

Transmission Header Receive

The following variables contain decoded information for received SNA information.

RDAF (-- address)
Contains the DAF (destination address field) located in bytes 2 and 3 of FID0 and FID1, and byte 2 of FID2. The field specifies which node the message is directed. Valid values are 0 through 65535 for FID0 and FID1, and 0 through 255 for FID2.

ROAF (-- address)
Contains the OAF (origination address field) located in bytes 4 and 5 of FID0 and FID1, and byte 3 of FID2. This field specifies the node transmitting the message. Valid values are 0 through 65535 for FID0 and FID1, and 0 through 255 for FID2.

RSNF (-- address)
Contains the SNF (sequence number field) located in bytes 6 and 7 of FID0 and FID1, bytes 4 and 5 of FID2, and bytes 22 and 23 of FID4. This field is generated by the LU (logical unit) and used for detection of lost messages. Valid values are 0 through 65535.

Transmission Header Transmit

S.FID–ID (-- address)
Contains the format identifier field of the FID2. As the BUILD_TH–BYTE0 or BYTE_TH commands are only supported for an FID2 header, the only permissible value for this variable is hex 20.

S.MAPPING–FIELD (-- address)
Contains the mapping field of the FID2.

ITL Constant	Description
R*WHOLE–BIU (default)	Entire segment
R*FIRST–SEGMENT	First segment
R*LAST–SEGMENT	Last segment
R*MIDDLE–SEGMENT	Middle segment

S.EXP-FLOW-IND (-- address)
Contains the expedited flow indicator field of the FID2.

ITL Constant	Description
R*NONE	Normal
R*EXPEDITED (default)	Expedited

SOAF (-- address)
Contains the OAF (origin address field) of the FID2. Valid values are 0 (default) through 255.

SDAF (-- address)
Contains the DAF (destination address field) of the FID2. Valid values are 0 (default) through 255.

SSNF (-- address)
Contains the SNF (sequence number field) of the FID2 when the format indicator of the request/response header, within the same frame, is set to 0. Valid values are 0 (default) through 65535.

SSNF1 (-- address)
Contains the SNF (sequence number field) of the FID2 when the format indicator of the request/response header, within the same frame, is set to 1. Valid values are 0 (default) to 65535.

Request/Response Header Transmit

S.RH-ID (-- address)
Contains the RU category field of the request/response header.

Hex Value	Description
00	FMD (function management data)
20	NC (network control - default)
40	DFC (data flow control)
60	SC (session control)

S.RH-REQ-RES (-- address)
Contains the request/response indicator.

Hex Value	Description
0	Request header (default)
80	Response header

S.FORMAT-IND (-- address)

Contains the format indicator field of the request/response header.

ITL Constant	LU-LU Session Description	SSCP Session Description
R*FORMAT0	No FM header follows	Character coded RU (request/response unit)
R*FORMAT1 (default)	FM header follows	Field formatted RU

S.SENSE-DATA-IND (-- address)

Contains the sense data indicator field of the request/response header.

ITL Constant	Description
R*NONE (default)	Sense data not included
R*SENSE-DATA-INCL	Sense data include (i.e. an error has occurred)

S.CHAIN-IND (-- address)

Contains the begin and end chain fields of the request/response header.

ITL Constant	Description
R*MIDDLE-RU	Middle RU in the chain
R*LAST-RU	Last RU in the chain
R*FIRST-RU	First RU in the chain
R*ONLY-RU	Only RU in the chain (default)

S.RESPONSE-TYPE (-- address)

Contains the DR1I, DR2I (definite response indicators 1 and 2), and the ERI (exception response indicator) for a request header. The definite response indicators 1 and 2 and the response type indicator for a response header.

ITL Constant	Request Header	Response Header
R*RQN	No response requested	Reserved
R*RQD2	Definite response requested by DR2I	Positive response to DR2I
R*RQE2	Exception response requested by DR2I	Negative response to DR2I
R*RQD1 (default)	Definite response requested by DR1I	Positive response to DR1I
R*RQE1	Exception response requested by DR1I	Negative response to DR1I
R*RQD3	Definite response requested by DR1I and DR2I	Positive response to DR1I and DR2I
R*RQE3	Exception response requested by DR1I and DR2I	Negative response to DR1I and DR2I

S.QUEUED-RESP-IND (-- address)

Contains the queued response indicator of the request/response header.

ITL Constant	Description
R*NONE (default)	Response bypasses transmission control queues
R*QUEUED	Enqueue response in transmission control queues

S.PACING-IND (-- address)

Contains the pacing indicator field of the request/response header.

ITL Constant	Description
R*NONE (default)	Pacing not requested
R*PACING	Pacing requested

S.BRACKET-IND (-- address)

Contains the begin and end bracket indicators of a request header.

ITL Constant	Description
R*NONE (default)	Between or within brackets
R*END	Notification to other half session that the speaker is finished and that the bracket protocol is complete
R*BEGIN	Request by a half session to become speaker of the session
	Only chain in the bracket

S.CHANGE-DIR-IND (-- address)

Contains the change direction indicator of the request header.

ITL Constant	Description
R*NONE (default)	No change in direction
R*CHANGE	Change direction

S.CODE-SEL-IND (-- address)

Contains the code selection indicator of the request header.

ITL Constant	Description
R*CODE0 (default)	Code 0
R*CODE1	Code 1

S.ENCIPH-DATA-IND (-- address)
Contains the enciphered data indicator of the request header.

ITL Constant	Description
R*NONE (default)	RU not enciphered
R*PADDED	RU enciphered

S.PADDED-DATA-IND (-- address)
Contains the padded data indicator of the request header.

ITL Constant	Description
R*NONE (default)	RU not padded
R*PADDED	RU was padded before encipherment

Sense Data Transmit

Sense data is inserted after the RH if the S.SENSE-DATA-IND variable contains a non-zero value.

S.SD-ID (-- address)
Contains the sense data category.

Hex Value	Description
0 (default)	User sense data only
08	Request reject
10	Request error
20	State error
40	RH (request header) usage error
80	Path error

S.SD-CODE (-- address)
Contains the sense data modifier and sense code specific information fields.

NOTE
The upper byte of this variable is not used when creating the sense data field with the BUILD_RH command.

Example:
Set the sense data category to request reject, sense data modifier to session limit exceeded, and fill in sense data code bytes with zero's.

```
8 S.SD-ID !                        ( Make sense data category to request reject )
 0X00050000 S.SD-CODE  !           ( Create sense data modifier and sense code fields )

Ignored  Sense   Sense
          Data   Code
        Modifier
```

NOTE
See Appendix A for defined sense data categories. Refer to IBM's System Network Architecture, Reference Summary, for more information.

Request/Response Unit Transmit

The following variable is used for building a request/response unit when SNA fields are constructed using the BUILD_RU command.

S.RU–ID (-- address)
Contains the response/request unit code. Valid ITL constants are shown in Tables 13–4 through 13–7.

12
EMULATION RESPONSE

The SDLC Emulation is implemented as a state–driven protocol emulation. Point–to–point and multipoint configuration is supported as well as two way alternate or two way simultaneous transmission. The emulation can be run as a primary station which controls the data link and issues commands to up to thirty–two secondary stations controlled by a primary. When acting as a secondary, the emulation receives commands and issues responses.

The emulation has been set up to run as:

- an automatic simulation which operates in accordance with IBM's SDLC procedure for modulo 8;
- a semi–automatic tester. The test manager is used to build and execute test scenarios to test responses and for generation of errors; and to implement SNA information within SDLC information frames;
- a manual tester. The test is controlled from the user's keyboard;
- up to 32 simultaneous links.

12.1 Emulation State Machines

To ensure correct protocol operation, state machines have been implemented. Based on input events (i.e. received frames and timers) transitions from one state to another are made in accordance with IBM's SDLC implementation. Up to 32 state machines are supported simultaneously (i.e. one state machine per link). Separate NR and NS counts are maintained for each link.

Different states are applicable for the primary and secondary ends of the emulation. Emulation states are defined by ITL constants, as shown in the following tables.

ITL Constant	Description
NO_LINK	No link is established with the secondary. No I frames can be sent
LINK	A link has been established and polling is underway.
SEND_DISC	The primary is waiting for the secondary to relinquish control of the link so that is can send a DISC.
DISC/UA_WAIT	The primary has sent a DISC and is waiting for a UA. If a UA is not received, another DISC is sent out.
SIM/UA_WAIT	The primary has sent an SIM and it is waiting for the secondary to respond with a UA.
SEND_SNRM	The primary is waiting for control of the link so it can send out an SNRM frame.
SNRM/UA_WAIT	The primary has sent an SNRM and is waiting for the secondary to respond with a UA. If a UA is not received, another SNRM is sent out.

Table 12–1 Primary Emulation States

ITL Constant	Description
NDM	The normally disconnected mode state. The secondary responds with a DM or RIM when it is polled.
NRM	The normal response mode state and indicates that a link has been established. If polled an RR or RNR is returned. This is the only state of the secondary state machine in which I frames can be sent.
FRAME_REJECT	The frame reject state. If the secondary encounters an invalid frame from the primary, it enters this state and will only return FRMR's. The emulation leaves this state when the primary sends an SIM, SNRM or DISC frame.

Table 12–2 Secondary Emulation States

Sometimes it is convenient to force the emulation into a specific state to observe the reaction of the DUT (device under test) under error conditions. The state machine can be forced into any state with the ENTER command.

ENTER (state --)

Forces the automatic SDLC Emulation into a new state. Use the desired primary or secondary state from the previous tables as an input parameter.

WARNING

The new state must belong to the same state machine set (i.e. primary or secondary) as unpredictable operation might result if a non-existent state is entered.

Example:

Force a primary SDLC Emulation into the NO_LINK state.

```
NO_LINK ENTER
```

12.2 Automatic Responses

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

The following commands activate and deactivate state machines.

EMUL_ON (--)
Activates the primary/secondary state machine (default). Automatic responses to received frames are generated.

EMUL_OFF (--)
Deactivates the primary/secondary state machine. No automatic responses to received frames are generated.

RUN_LAYER2 (--)
Forces processing of frames and timeouts through the emulation if the emulation has been turned off.

12.3 State–Dependent Send Commands

Either function keys or commands are used to transmit frames 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 protocol state machine must be activated (see Section 14 for examples).

FORM_LINK (--)
A primary emulation command; if issued on a secondary, the command is ignored. The action of this command depends upon whether or not a link has been established previously.

If a link has not been established previously (NO_LINK state), the primary station:
- sends out a broadcast XID (PU address hex FF); and
- expects to receive an XID from each secondary PU address defined on the Station Setup Menu (up to 32 supported).

On the currently selected station:
- when an XID is received, the primary responds with an RR and expects an RR in return; and
- when the RR is received, the primary sends out an SNRM and expects a UA (SNRM/UA–WAIT state). When the UA is received, polling starts (LINK state).

On all other supported stations:
- when an XID is received, the primary responds with a DISC and expects a DM in response. There is no change in state.

If a link has previously been established on any PU, the primary station:

- sends out an SNRM frame on the currently selected PU address and goes to the SNRM/UA_WAIT state; and
- once the expected UA is received, the emulation goes to the LINK state and polling (transmission of RR frames) starts on the currently selected PU address.

NOTE
The link should be formed separately for each PU address with which the primary station intends to communicate.

Send topic
Enter PU function key
Form Link function key

INIT_MODE (--)

A secondary emulation command; if issued on a primary, the command is ignored. The result of this command depends upon whether or not a link is up.

If a link has been established by the primary on the currently selected PU, the secondary station using this PU (address):

- sends out an RIM and expects an SIM in response;
- responds with a UA after reception of the SIM;
- responds with a UA after the primary transmits an SNRM; and
- starts polling (transmission of RR frames between the primary and secondary).

If a link is not established by the primary on the currently selected PU, the secondary emulation takes no action.

Send topic
Initialize Link function key

KILL_LINK (--)

This command can be issued on either a primary or secondary emulation. If a link is not established on the currently selected PU, the command is ignored.

If any link has been established and this command is issued on a primary emulation, the primary station:

- sends out a DISC frame on this PU and goes to the DISC/UA_WAIT state. Once this expected UA is received, polling on this PU ceases.

If a link has been established and this command is issued on a secondary emulation, the secondary station:

- sends out an RD frame on the currently selected PU and expects to receive a DISC frame on this PU in response;
- responds to the DISC with a UA frame;
- responds with a DM frame if an RR frame is received on this PU;
- responds with a UA frame if another SNRM is received on this PU, the link is re–established, and polling continues.

Send topic
Kill Link function key

SNRM (--)

A primary emulation command; if issued on a secondary, the command is ignored.

If a link has not been established previously, the primary station:

- remains in a NO_LINK state and no action is taken.

If a link has previously been established, the primary station:

- is forced into a NO_LINK state;
- resets the values of N(S) and N(R) to 0 for this PU and an SNRM is sent; and
- is then forced into the SNRM/UA_WAIT state and expects to receive a UA frame.

DISC (--)

A primary emulation command; if used on a secondary, this command is ignored.

If a link has never been established, the emulation remains in a NO_LINK state and no action is taken.

If a link has been established and the primary has control of the link, the primary sends a DISC and enters the DISC/UA_WAIT state for this link.

If a link has been established and the secondary has control of the link, the primary is forced into the SEND_DISC state, i.e. it is waiting for the secondary to relinquish control of the link so that it can send a DISC.

SIM (--)

A primary emulation command; if issued on a secondary, the command is ignored.

If a link has never been established, the primary station:

- sends a broadcast SIM with a PU address of hex FF and enters the SIM/UA_WAIT state;
- sends out an SNRM when a UA is received on the current PU, and goes to the SNRM/UA_WAIT state;
- sends a DISC frame when a UA is received on all other PU's;
- sets the N(R) and N(S) values to 0 when the emulation receives a UA from the current PU and the emulation goes to the LINK state;
- sends an XID frame on the current PU; and
- starts polling on this PU when an XID frame is received on this PU.

If a link has been established, the primary station:

- is forced into the NO LINK state;
- sends out an SNRM on the current PU and goes to the SNRM/UA_WAIT state;
- transmits an SIM on the current PU and goes to the SIM/UA_WAIT state;
- returns to the SNRM_UA state when a UA is received and sends out an SNRM on the current PU; and
- goes to the LINK state after receiving a UA, and polling starts or continues on this PU.

XID (--)

A primary emulation command; if issued on a secondary, the command is ignored.

If a link has never been established via FORM_LINK; the command is ignored.

If a link has been established but is currently down:
- the emulation sends out an XID on the current address and expects an XID to be returned on the same PU.

If the emulation is in the LINK state, the primary station:
- sends out an XID on the current PU; and
- when it receives an XID on the same PU, polling resumes.

GET_ID (--)

A primary emulation command; if issued on a secondary, the command is ignored.

If a link has been established, an XID exchange is initiated.

RIM (--)

A secondary emulation command; if issued on a primary, the command is ignored.

If a link has never been established or if the currently selected link is down, the secondary remains in the NDM state and ignores the command.

If the link is up, the secondary station:
- is forced into NDM state;
- sends an RIM frame;
- sends a UA frame when an SIM frame is received from the primary; and
- responds with a UA frame if the secondary then receives an SNRM frame and goes to the NRM state and is ready to reply with an RR or RNR if polled.

RD (--)

A secondary emulation command; if issued on a primary, the command is ignored.

If a link has never been established or the currently selected link is down, the secondary remains in the NDM state and ignores the command.

If the link is up, the secondary station:
- transmits an RD at the first opportunity and waits for a DISC frame; and
- sends a UA frame when it receives a DISC frame from the primary and goes to the NDM state.

POLL_IN_I_FRAME (--)

Sends the last information frame in the window with the poll bit set to 1 (default).

NO_POLL_IN_I_FRAME (--)

Sends all information frames with the poll bit equal to 0. Following transmission of the last information frame in the window, an RR frame is sent with the poll bit set to 1, thus requesting acknowledgment.

OK (--)
Once a link has been established, I frames can be accepted and the emulation sends out RR frames in response to a poll (default).

BUSY (--)
Once a link has been established, no I frames can be accepted and the emulation sends out RNR frames in response to a poll.

RESET (--)
If used with a primary emulation, resets all parameters and unconditionally terminates the link. Any unsent frames are cancelled.

If used with a secondary emulation:

- all parameters are reset;
- a DM frame is transmitted; and
- if a DISC frame is received, the link is terminated.

The following two commands transmit user-defined data. Up to 80 characters are allowed when directly input from the keyboard and 255 characters within test scripts.

SENDF (string --)
Transmits a user-defined frame.

Example:
Transmit an SNRM with P bit set on PU address of hex BB.

```
X" BB93" SENDF
```

SEND_BTU (string --)
Transmits a user-defined information field BTU (basic transmission unit) in an I frame. This I frame is sent over the current link with the correct N(R), N(S) and address of that link.

Example:
Send out an ACTLU request within all FID2 transmission header.

```
X" 2D0002000B536B80000D0101" SEND_BTU
```

X" $\underbrace{\text{2D0002000B53}}_{\text{FID2 TH}}\underbrace{\text{6B8000}}_{\text{Request RH}}\underbrace{\text{0D0101}}_{\text{ACTLU RU}}$"

The following four commands are used to set fields within an FID2. They must not be used to set these fields in other types of transmission headers. These commands are used in conjunction with SEND_BTU to transmit a frame containing the modified transmission header. The string must be created with dummy information in the respective fields.

SET_DAF (string\DAF value --string)
The string contains the correct number of bytes (6) for an FID2 plus the request/response header, request/response unit and sense data if desired. SET_DAF takes the specified DAF value and places it in the destination address field location of the specified string. Valid values for DAF within FID2 are 0 through 255.

Example:
Transmit a frame containing an FID2 transmission header with a predefined DAF, a response header and an ACTPU response unit.

```
X" 2D0000000B52EB800011020105000000000015"      ( Define string )SDAF @
( Obtain last defined SDAF )
SET_DAF                                          ( Modify string )
SEND_BTU                                         ( Send frame )
```

```
                DAF
X" 2D0000000B52EB80001 1020105000000000015"
      FID2 TH    Response      ACTPU RU
                 Header
```

SET_OAF (" string"\OAF value --" string")
The string contains the correct number of bytes (6) for an FID2 plus the request/response header, request/response unit and sense data if desired. SET_OAF takes the specified OAF value and places it in the origin address field location of the specified string. Valid values for OAF are 0 through 255.

Example:
Transmit a frame containing an FID2 with a predefined OAF, a request header and an ACTLINK request unit.

```
X" 2D0002000B536B800001020A0102"     ( Define string )
SOAF @                               ( Obtain predefined OAF )
SET_OAF                              ( Modify string )
SEND_BTU                             ( Send frame )
```

```
          OAF
X" 2D0002000B536B800001020A0102"
     FID2 TH     Request   ACTLINK R4
                 Header
```

SET_SNF (string\SNF value -- string)

The string contains the correct number of bytes for an FID2 plus the request/response header, request/response unit and sense data if required. SET_SNF takes the specified SNF value and places it in the sequence number field location of the specified string. Valid values for SNF are 0 through 65535.

Example:

Transmit a frame containing an FID2 with a predefined SNF; and a response header with sense data indicating a request reject category with the session limit exceeded.

```
X" 2D0000000B52EF900008050000"     ( Define string )
SSNF @                             ( Obtained predefined SNF )
SET_SNF                            ( Modify string )
SEND_BTU                           ( Send frame )
```

SNF

X" 2D0000000B52EF900008050000"

FID2 TH

Response Header Indicating Sense Data

Sense Data

RETURN_FIELDS (string--string)

The string contains the correct number of bytes for an FID2 plus the request/response header, request/response unit, and sense data if desired.

This command:

- takes the last received DAF and places it in the origin address field location of the specified string;
- takes the last received OAF and places it in the destination address field location of the specified string; and
- takes the last received SNF and places it in the sequence number field of the specified string.

The modified string is left as an output parameter.

Example:
Transmit a frame with an FID2 transmission header with the OAF, DAF, and SNF of the last received frame echoed back and the received OAF and DAF swapped and used as the DAF and OAF of the transmitted frame.

```
X" 2D0000000B52EB800011020105000000000015"     ( Define string )
RETURN_FIELDS                                  ( Swap and echo fields )
SEND_BTU                                       ( Send frame )
```

```
              DAF OAF SNF
X" 2D0000000B52EB800011020105000000000015"
          FID2      Response         ACTPU RU
                       RH
```

The following four commands are used to construct SNA fields within an FID2 header. They can be used in conjunction with the SET_LENGTH command. To transmit a frame after the SNA fields have been constructed, the SEND_BTU command is used. See Section 11 for a description of variables and valid values used in construction.

BUILD_TH-BYTE0 (" string "--" string")
Forms byte 0 of the transmission header by modifying the first byte of the specified string.

S.FID-ID	- Only FID2 valid
S.MAPPING-FIELD	- Mapping field
S.EXP-FLOW-IND	- Expedited flow indicator

Example:
Build byte 0 of an FID2 with a mapping field indicating a first segment and normal flow.

```
0x20  S.FID-ID !                                       ( FID2 )
R*FIRST-SEGMENT  S.MAPPING-FIELD !                     ( First segment )
R*NONE  S.EXP-FLOW-IND !                               ( Normal flow )
X" 0000000000000000000000000000000000000000"           ( Define string )
BUILD_TH-BYTE0                                         ( Build byte 0 )
```

Transmit the frame:

```
SET_LENGTH          ( Sets length of TH i.e. in this case 6 bytes )
SEND_BTU            ( Send frame )
```

NOTE
In the previous example, SET_LENGTH limits the length of the information field transmitted to six bytes, i.e. only the FID2.

BUILD_TH (string--string)

Builds an FID2 by modifying the first six bytes of the string reflecting the values in the following variables. The modified string is left as an output parameter.

S.FID-ID	- Only FID2 valid
S.MAPPING-FIELD	- Mapping field
S.EXP-FLOW-IND	- Expedited flow indicator
SOAF	- Origin address field
SDAF	- Destination address field
S.FORMAT-IND	- Format indicator from request/response header
SSNF	- Sequence number field
or	
SSNF1	- Sequence number field for format 1, expedited flow

Example:
Build an FID2 with a mapping field indicating a whole segment, expedited flow, an origin address field of four, a destination address field of six, and format 1.

```
0x20  S.FID-ID !                        ( FID2 )
R*WHOLE-BIU  S.MAPPING FIELD !          ( Whole segment )
R*EXPEDITED  S.EXP-FLOW-IND !           ( Expedited flow )
4 SOAF !                                ( Origin address field )
6 SDAF !                                ( Destination address field )
R*FORMAT1  S.FORMAT-IND !               ( Format 1 )
X" 000000000000000000000000000000000000000000"     ( Define string )
BUILD_TH                                ( Build transmission header )
```

Transmit the frame.

```
SET_LENGTH                              ( Sets length of transmission header )
SEND_BTU                                ( Send in the frame )
```

NOTE
In this example, SET_LENGTH limits the length of the information field transmitted to 6 bytes, i.e. only the FID2.

BUILD_RH (" string "--" string")

Builds a request/response header by modifying the seventh to ninth bytes of the string reflecting the values in the following variables. The modified string is left as an output parameter.

S.RH-ID	- RU category
S.RH-REQ-RES	- Request or response indicator
S.FORMAT-IND	- Format indicator
S.SENSE-DATA-IND	- Sense data indicator
S.CHAIN-IND	- Chain indicator (begin and end)
S.RESPONSE-TYPE	- Response indicator
S.QUEUED-RESP-IND	- Queued response indicator
S.PACING-IND	- Pacing indicator
S.BRACKET-IND	- Bracket indicator (begin and end)
S.CHANGE-DIR-IND	- Change direction indicator
S.CODE-SEL-IND	- Code selection indicator
S.ENCIPH-DATA-IND	- Enciphered data indicator
S.PADDED-DATA-IND	- Padded data indicator

Example:
Build a session control request header.

```
0X60 S.RH-ID !                    ( Session control )
0 S.RH-REQ-RES !                  ( Request header )
R*FORMAT1 S.FORMAT-IND !          ( Format 1 )
R*NONE S.SENSE-DATA-IND !         ( No sense data )
R*ONLY-RU S.CHAIN-IND !           ( Only message in chain )
R*RQD1 S.RESPONSE-TYPE !          ( Definite response requested by DR1 )
R*NONE S.QUEUED-RESP-IND !        ( Transmission control queues bypassed )
R*NONE S.PACING-IND !             ( Pacing not requested )
R*NONE S.BRACKET-IND !            ( Between or within brackets )
R*NONE S.CHANGE-DIR-IND !         ( No direction change )
R*CODE0 S.CODE-SEL-IND !          ( Code 0 selected )
R*NONE S.ENCIPH-DATA-IND !        ( RU not enciphered )
R*NONE S.PADDED-DATA-IND !        ( RU not padded )
X" 2D0000000B5200000000000000000000000000"     ( Define string )
BUILD_RH
```

Transmit the frame.

```
SET_LENGTH    ( Sets the length of transmission header and request header )
SEND_BTU      ( Send the frame )
```

NOTE
*In this example, SET_LENGTH limits the length of the information field transmitted to 9 bytes. If the S.SENSE-DATA-IND variable contained R*SENSE-DATA-INCL, the length would be 13 bytes to include 4 bytes of sense data.*

BUILD_RU (string--string)
Builds a request/response unit by modifying the appropriate location in the specified string using the following variables. The modified string is left as the output parameter.

S.RU-ID	Response/request unit code
S.SENSE-DATA-IND	Sense data indicator

Example:
Build a complete frame using BUILD_TH, BUILD_RH and BUILD_RU as described in previous examples.

```
R*FID2 S.FID-ID !                       ( FID2 )
R*WHOLE-BIU S.MAPPING-FIELD !           ( Whole segment )
R*EXPEDITED S.EXP-FLOW-IND !            ( Expedited flow )
4 SOAF !                                ( Origin address field )
6 SDAF !                                ( Destination address field )
R*FORMAT1 S.FORMAT-IND !                ( Format 1 )
X" 000000000000000000000000000000000000000000"
BUILD_TH                                ( Build transmission header )
R*SC S.RH-ID !                          ( Session control )
R*REQ-HDR S.RH-REQ-RES !                ( Request header )
R*FORMAT1 S.FORMAT-IND !                ( Format 1 )
R*NONE S.SENSE-DATA-IND !               ( No sense data )
R*ONLY-RU S.CHAIN-IND !                 ( Only message in chain )
R*RQD1 S.RESPONSE-TYPE !                ( Definite response requested by DR1 )
R*NONE S.QUEUED-RESP-IND !              ( Transmission control queues bypassed )
R*NONE S.PACING-IND !                   ( Pacing not requested )
R*NONE S.BRACKET-IND !                  ( Between or within brackets )
R*NONE S.CHANGE-DIR-IND !               ( No direction change )
R*CODE0 S.CODE-SEL-IND !                ( Code 0 Selected )
R*NONE S.ENCIPH-DATA-IND !              ( RU not enciphered )
R*NONE S.PADDED-DATA-IND !              ( RU not padded )
BUILD_RH                                ( Build request header )
R*ACTLU S.RU-ID !                       ( ACTLU request code )
BUILD_RU                                ( Build response unit )
```

Transmit this frame.

```
SET_LENGTH                              ( Set length of frame )
SEND_BTU                                ( Send frame )
```

NOTE
In this example, SET_LENGTH limits the length of the information field to 12 bytes, 6 for the FID2, 3 for the request header, and 3 for the request unit.

SET_LENGTH (string--string)
This string is count prefixed, i.e. contains a count of the number of characters in the string. SET_LENGTH stores the length of the SNA fields (TH, RH, SD and RU) in this count.

WARNING
Care must be used in using the SET_LENGTH command.

- *If none of BUILD_TH-BYTE0, BUILD-TH, BUILD-RH, or BUILD_RU have been used since program startup, the length of the SNA fields is set to zero.*
- *If BUILD_TH-BYTE0 or BUILD_TH was the last SNA field commands to be called, the length of the SNA fields is set to 6.*
- *If BUILD_RH was the last SNA field command to be called, the length of the SNA fields is set to 9 if no sense data is included or 13 if sense data is included.*
- *If BUILD_RU was the last SNA field command to be called, the length of the SNA fields is set to include the FID2 transmission header, the request/response header, sense data if included and the request/response code.*

NOTE
To change the length of the SNA fields after calling SET_LENGTH or building the fields, use:
N OVER C!
This presumes the string is on the stack and has a length of at least N characters.

Example:
Build the SNA fields as shown under BUILD_RU. Limit the length of the SNA fields to 10 bytes. The modified string created in the example remains on the stack for transmission with SEND_BTU.

```
X" 2D0000000B5200000000000000000000000000"     ( Define string )
BUILD_RH
SET_LENGTH        ( Sets the length of transmission header and request header )
10 OVER C!                                     ( Set length of SNA fields to 10 )
SEND_BTU                                       ( Send the frame )
```

12.4 State-Independent Send Commands

The following commands transmit frames without regard to correct protocol procedure. They do not update the protocol states or increment sequence values, i.e. N(R) AND N(S). They are designed to be used in test scenarios when frames are generated out of context. The variables used in these commands are described in Section 11.

S:SNRM (--)
Transmits an SNRM command frame using values from the following variables.
ID-OUT - Address of the current link (valid values are 0 through 255)
P/F_BIT - P/F bit to be transmitted (0 or 1)

S:UA (--)
Transmits a UA response frame using values from the following variables.
ID-OUT - Address of the current link (valid values are 0 through 255)
P/F_BIT - P/F bit to be transmitted (0 or 1)

S:RD (--)
Transmits an RD response frame using values from the following variables.
ID-OUT - Address of the current link (valid values are 0 through 255)
P/F_BIT - P/F bit to be transmitted (0 or 1)

S:DISC (--)
Transmits a DISC command frame using values from the following variables.
ID-OUT - Address of the current link (valid values are 0 through 255)
P/F_BIT - P/F bit to be transmitted (0 or 1)

S:RIM (--)
Transmits an RIM response frame using values from the following variables.
ID-OUT - Address of the current link (valid values are 0 through 255)
P/F_BIT - P/F bit to be transmitted (0 or 1)

S:SIM (--)

Transmits an SIM command frame using values from the following variables.

ID-OUT	– Address of the current link (valid values are 0 through 255)
P/F_BIT	– P/F bit to be transmitted (0 or 1)

S:DM (--)

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

ID-OUT	– Address of the current link (valid values are 0 through 255)
P/F_BIT	– P/F bit to be transmitted (0 or 1)

S:UP (--)

Transmits an UP command frame using values from the following variables.

ID-OUT	– Address of the current link (valid values are 0 through 255)
P/F_BIT	– P/F bit to be transmitted (0 or 1)

S:FRMR (--)

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

ID-OUT	– Address of the current link (valid values are 0 through 255)
P/F_BIT	– P/F bit to be transmitted (0 or 1)
CONTROL	– Value contained in the control field of the last received frame. See Tables A-1 and A-2 for defined values.
NR	– Value of N(S) expected in the next received numbered I frame (valid values are 0 through 7).
NS	– Value of N(S) in the next numbered I frame to be sent (valid values are 0 through 7).
BAD-CONTROL	– Constant value of 1. When the frame is passed through the SNA Monitor, this is interpreted as an invalid frame control field.

S:CFGR (--)

Transmits a CFGR frame using values from the following variables.

ID-OUT	– Address of the current link (valid values are 0 through 255)
P/F_BIT	– P/F bit to be transmitted (0 or 1)

S:BCN (--)

Transmits a BCN response frame using values from the following variables.

ID-OUT	– Address of the current link (valid values are 0 through 255)
P/F_BIT	– P/F bit to be transmitted (0 or 1)

S:TEST (--)

Transmits a TEST frame with user-defined text using values from the following variables.

ID-OUT	– Address of the current link (valid values are 0 through 255)
send_test	– Buffer used for sending user-defined test message of up to 63 characters

Example:
Transmit a TEST frame containing the hex values of 01020304 in the test text field.

```
X" 01020304"                  ( Define TEST text )
COUNT 63 MIN DUP              ( Obtain count up to 63 characters )
send_test C!                  ( Store count in first byte of buffer )
send_test 1+ SWAP CMOVE       ( Store text in buffer starting at second byte )
S:TEST                        ( Send TEST frame )
```

S:UI (--)

Transmits a UI frame with a user-defined information field using values from the following variables.

ID-OUT	– Address of the current link (valid values are 0 through 255)
P/F_BIT	– P/F bit to be transmitted (0 or 1)
send_ui	– Buffer used for sending user-defined information field in UI frames (up to 63 characters).

Example:
Send a UI frame containing the hex value 01020304 in the information field and a P bit equal to zero.

```
0 P/F_BIT !                ( Set P bit to zero )
X" 01020304"               ( Define text for information field )
COUNT 63 MIN DUP           ( Obtain count of up to 63 characters )
send_ui C!                 ( Store count in first byte of buffer )
send_ui 1+ SWAP CMOVE      ( Store text in buffer starting at second byte )
S:UI                       ( Send UI frame )
```

S:I (--)

Transmits a numbered I frame with a user-defined information field using values from the following variables.

NR	– Value of N(S) expected in next received numbered I frame (valid values are 0 through 7).
NS	– Value of N(S) in the I frame being sent (valid values are 0 through 7).
ID-OUT	– Address of the current link (valid values are 0 through 255)
send_info	– Buffer used for sending user-defined information field (up to 63 characters) in I frames.

Example:
Send an I frame containing the hex value 01020304 in the information field.

```
X" 01020304"               ( Define text for information field )
COUNT 63 MIN DUP           ( Obtain count of up to 63 characters )
send_info C!               ( Store count in first byte of buffer )
send_info 1+ SWAP CMOVE    ( Store text in buffer )
S:I
```

S:RR (--)

Transmits an RR frame using values from the following variables.

ID-OUT	– Address of the current link (valid values are 0 through 255)
P/F_BIT	– P/F bit to be transmitted (0 or 1)
NR	– Value of N(S) expected in next received numbered I frame (valid values are 0 through 7).

S:RNR (--)

Transmits an RNR frame using values from the following variables.

ID-OUT	– Address of the current link (valid values are 0 through 255)
P/F_BIT	– P/F bit to be transmitted (0 or 1)
NR	– Value of N(S) expected in next received numbered I frame (valid values are 0 through 7).

S:REJ (--)

Transmits an REJ frame using values from the following variables.

ID-OUT	- Address of the current link (valid values are 0 through 255)
P/F_BIT	- P/F bit to be transmitted (0 or 1)
NR	- Value of N(S) expected in next received numbered I frame (valid values are 0 through 7).

12.5 CRC Errors

Frames can be sent with correct or incorrect CRC's (FCS), no CRC, or aborted during the transmission.

CRC-TYPE (-- address)

Contains the error type status used by the current link when transmitting frames.

ITL Constant	Description
CRC (default)	A normal CRC is appended
CRC_ERR	A bad CRC is used
NO_CRC	No CRC is attached to the frame
ABT	The frame is aborted part way through

Example:

Force the current link to transmit CRC errors with each frame sent.

```
CRC_ERR CRC-TYPE !
```

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 SNA Monitor/SDLC Emulation applications.

For a complete explanation of the test manager and tools available, see the Programmer's Reference Manual.

This section review basic ITL components and describes the protocol event and action commands specific to the SNA Monitor and SDLC Emulation.

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 (n --)
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 (n --)
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.

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-specified of the first action within that state. If the evaluation does not return a true value, 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.

Layer 1

If the SNA Monitor or SDLC Emulation is running on a B-Channel 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.

Received Frames

ITL provides recognition of protocol specific frames, SNA protocol units, 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-ID variable is associated with each frame or BLU (basic link unit).

Frame (BLU) Identifier	Description
R*RR	Receive ready
R*RNR	Receive not ready
R*REJ	Reject
R*SNRM	Set normal response mode
R*DISC	Disconnect
R*SIM	Set initialization mode
R*UI	Unnumbered information
R*TEST	Test
R*XID	Exchange information
R*UA	Unnumbered acknowledgment
R*RD	Request disconnect
R*RIM	Request initialization mode
R*DM	Disconnect mode
R*UP	Unnumbered poll
R*FRMR	Frame reject
R*I	Numbered information present
R*SNRME	Set normal response mode eExtended
R*BCN	Beacon
R*CFGR	Configure
R*SHORTFRM	Short frame (no control field)
R*INVFRM	Invalid frame
R*INVCRC	Invalid CRC

Table 13–1 Frame (BLU) Identifiers

?RX_BLU (frame identifier -- flag)
Returns true if the specified frame is received. See Table 13–1 for a list of valid frame identifiers.

Example:
Check for reception of an RR frame.

```
R*RR ?RX_BLU
```

Example:
Check for reception of any supervisory frame (a logical 'OR' can be used).

```
1 STATE{
    R*RR ?RX_BLU
    R*RNR ?RX_BLU  OR
    R*REJ ?RX_BLU  OR                         ( Supervisory frame received? )
    ACTION{
        T." Supervisory Frame received." TCR   ( Create trace statement )
        2 NEW_STATE
     }ACTION
  }STATE
```

?RX_FRAME (frame id#1\...\frame id#n\n -- flag)

Returns true if one of the specified frames is received. See Table 13-1 for a list of valid frame identifiers.

Example:
Check for reception of any supervisory frame.

```
1 STATE{
      R*RR R*RNR R*REJ 3 ?RX_FRAME            ( Supervisory frame received? )
      ACTION{
          T." Supervisory Frame received."    ( Create trace statement )
          TCR
          2 NEW_STATE
      }ACTION
  }STATE
```

NOTE
This example has the same effect as the first example shown under ?RX_BLU.

?ADDRESS (PU address -- flag)

Returns true if the specified PU address is the first byte of the received frame. Valid values are 0 through 255.

Example:
Check for reception of any frame with a PU address of 35.

```
35 ?ADDRESS
```

NOTE
This command can be used in conjunction with ?RX_BLU or ?RX_FRAME to screen out similar frames (BLU's).

Example:
Check whether an SNRM is received with the PU address of 200.

```
200 ?ADDRESS R*SNRM ?RX_BLU  AND
```

Example:
Check whether an SNRM or DISC is received with the PU address of hex 50.

```
0x50 ?ADDRESS
R*SNRM R*DISC 2 ?RX_FRAME  AND
```

FRAME? (-- flag)

Returns true if any frame is received.

DTE? (-- flag)

Returns true if any frame is received from a DTE. This command can be used with a logical 'AND' in combination with other frame event recognition commands.

Example:
Check whether an SNRM is received with a PU address of 10 from a DTE.

```
R*SNRM ?RX_BLU
10 ?ADDRESS AND
DTE?  AND
```

Example:
Check whether an SNRM or DISC is received with a PU address of 15 from a DTE.
```
R*SNRM R*DISC 2 ?RX_FRAME
15 ?ADDRESS AND
DTE?  AND
```

DCE? (-- flag)
Returns true if any frame is received from a DCE. This command can be used with a logical 'AND' in combination with other frame event recognition commands.

Example:
Check whether an SNRM is received with a PU address of 10 from a DCE.
```
R*SNRM ?RX_BLU 10 ?ADDRESS AND DCE?  AND
```

Example:
Check whether an SNRM or DISC is received with a PU address of 15 from a DCE.
```
R*SNRM R*DISC 2 ?RX_FRAME 15 ?ADDRESS AND DCE?  AND
```

ABORT? (-- flag)
Returns true if an abort frame is received.

CRC_ERROR? (-- flag)
Returns true if a frame with a CRC error is received.

QEMPTY? (queue id -- flag)
Returns true if the specified queue is empty. Any I frames or UI frames to be transmitted by the emulation are queued and sent when possible. Once a UI frame is transmitted, it is removed from the queue and unavailable for retransmission. When an I frame is sent, it is maintained in a backup queue until it is acknowledged. Each data link maintains its own separate queue.

Queue ID	Description
UI_QUEUE	UI queue
I_QUEUE	I queue
I_BACKUP	I frame backup queue

Example:
Check to see whether an I frame can be transmitted (i.e. is the I queue empty).
```
I_QUEUE QEMPTY?
IF
    ( Send frame here )
ENDIF
```

Transmission Header Events

A format identifier stored in the FID-ID variable is associated with each TH (transmission header).

Format Identifier	Description
R*FID0	FID0 transmission header
R*FID1	FID1 transmission header
R*FID2	FID2 transmission header
R*FID3	FID3 transmission header
R*FID4	FID4 transmission header
R*FIDF	FIDF transmission header
R*INVFID	Invalid transmission header

Table 13–2 Transmission Header Identifiers

NOTE
If the received frame does not contain a transmission header, the FID–ID variable contains the value 0.

Example:
Check for reception of an FID4 transmission header.
```
FID-ID @ R*FID4 =
```

?THAF (address field -- flag)
Returns true if the specified destination or origin address field is found in the transmission header of a received frame. ?THAF is available in FID0, FID1, and FID2 transmission headers. Valid values are 0 through 65535 for an FID0 or FID1, and 0 through 255 for an FID2.

Example:
Check for reception of a frame with either a destination or origin address field of 100 in FID0, FID1, and FID2.
```
100 ?THAF
```

Example:
Check for reception of an FID2 transmission header with either a destination or origin address field of 100.
```
FID-ID @ R*FID2 =
100 ?THAF  AND
```

?MAPPING_FIELD (mapping field -- flag)

Returns true if the specified mapping field is found in the transmission header. The mapping field is a two bit field present in FID0, FID1, FID2, FID3, and FID4, but not FIDF.

ITL Constant	Description
R*WHOLE-BIU	Entire message
R*FIRST-SEGMENT	First segment in message
R*LAST-SEGMENT	Last segment in message
R*MIDDLE-SEGMENT	Middle or intermediate segment in message

Example:
Check for reception of a transmission header containing a mapping field indicating the first segment in a message in FID0, FID1, FID2, FID3, and FID4.

```
R*FIRST-SEGMENT ?MAPPING_FIELD
```

Example:
Check for reception of an FID2 with a mapping field indicating the last segment in a message.

```
FID-ID @ FID2 =
R*LAST-SEGMENT ?MAPPING-FIELD  AND
```

Request/Response Header Events

An identifier stored in the RH-ID variable is associated with each RH (request/response header).

RU Category	Description
R*FMD	Function management data, network services
R*NC	Network control
R*DFC	Data flow control
R*SC	Session control
R*INVRH	Invalid RU category

Table 13-3 Request/Response Unit Category Identifiers

?RH_REQ (-- flag)

Returns true if the received frame contains a request header.

?RH_RES (-- flag)

Returns true if the received frame contains a response header.

?RX_RH (RU category -- flag)

Returns true if the specified RU category is found in the received frame. See Table 13-3 for valid RU categories.

Example 1:
Check for reception of a frame with an RU category of FMD (function management data).
```
R*FMD ?RX_RH
```

Example 2:
Check for reception of a frame containing a response header and an RU category of FMD.
```
?RH_RES R*FMD ?RX_RH  AND
```

Request/Response Unit Events

Protocol defined request/response units are associated with each category of RU.

?RX_RU (RU value -- flag)

Returns true if the specified request/response unit is found in the received frame. Valid RU values are listed in Table 13-4 to 13-7.

Example:
Check for reception of a frame containing an RU of abandon connection.
```
R*ABCONN ?RX_RU
```

The following table shows defined ITL constants for the function management data, network services RU category.

ITL Constant	Description
R*ABCONN	Abandon connection
R*ABCONNOUT	Abandon connect out
R*ACTCONNIN	Activate connection in
R*ACTLINK	Activate link
R*ACTTRACE	Activate trace
R*ADDLINK	Add link
R*ADDLINKSTA	Add link station
R*ANA	Assign network addresses
R*BINDF	Bind failure
R*CDCINIT	Cross-domain control initiate
R*CDINIT	Cross-domain initiate
R*CDSESSEND	Cross-domain session end
R*CDSESSSF	Cross-domain session setup failure
R*CDSESSST	Cross-domain session started
R*CDSESSTF	Cross-domain session takedown failure
R*CDTAKED	Cross-domain takedown
R*CDTAKEDC	Cross-domain takedown complete
R*CDTERM	Cross-domain terminate

Table 13-4 FMD Request/Response Unit Identifiers

ITL Constant	Description
R*CESLOW	Contents of FMD header = X" 01020C"
R*CEXLOW	Contents of FMD header = X" 01020D"
R*CINIT	Control initiate
R*CLEANUP	Clean up session
R*CONNOUT	Connect out
R*CONTACT	Contact
R*CONTACTED	Contacted
R*CTERM	Control terminate
R*DACTCONNIN	Deactivate connect in
R*DACTLINK	Deactivate link
R*DACTTRACE	Deactivate trace
R*DELETENR	Delete network resource
R*DELIVER	Deliver
R*DISCONTACT	Discontact
R*DISPSTOR	Display storage
R*DSRLST	Direct search list
R*DUMPFINAL	Dump final
R*DUMPINIT	Dump initial
R*DUMPTEXT	Dump text
R*ECHOTEST	Echo test
R*ER–INOP	Explicit route inoperative
R*ER_TESTED	Explicit route tested
R*ESLOW	Entering slowdown
R*EXECTEST	Execute test
R*EXSLOW	Exiting slowdown
R*FNA	Free network address
R*FORWARD	Forward
R*INIT–OTHER	Initiate–other
R*INIT–OTHER–CD	Initiate–other cross–domain
R*INIT–SELF_F0	Initiate–self format 0
R*INIT–SELF_F1	Initiate–self format 1
R*INITPROC	Initiate procedure
R*INOP	Inoperative
R*IPLFINAL	IPL final
R*IPLINIT	IPL initial
R*IPLTEXT	IPL text
R*ISETCV	Contents of FMD header = X" 410222"
R*LCP	Lost control point
R*LOREQD	Load required
R*NMVT	Network management vector transport
R*NOTIFY_SSCP–LU	Notify SSCP → LU
R*NOTIFY_SSCP–SSCP	Notify SSCP → SSCP

Table 13–4 FMD Request/Response Unit Identifiers [continued]

ITL Constant	Description
R*NS-IPL-ABORT	NS IPL abort
R*NS-IPL-FINAL	NS IPL final
R*NS-IPL-INIT	NS IPL initial
R*NS-IPL-TEXT	NS IPL text
R*NS-LSA	NS lost subarea
R*NSLSA	Contents of FMD header = 410285
R*NSPE	NS procedure error
R*PROCSTAT	Procedure status
R*RECFMS	Record formatted maintenance statistics
R*RECMD	Contents of FMD header = 010480
R*RECMS	Record maintenance statistic
R*RECSTOR	Record storage
R*RECTD	Record test data
R*RECTR	Record test results
R*RECTRD	Record trace data
R*REQACTLU	Request activate logical unit
R*REQCONT	Request contact
R*REQDISCONT	Request discontact
R*REQECHO	Request echo test
R*REQFNA	Request free network address
R*REQMS	Request maintenance statistics
R*REQTEST	Request test procedure
R*RNAA	Request network address assignment
R*ROUTE-TEST	Route test
R*RPO	Remote power off
R*SESSEND	Session end
R*SESSST	Session started
R*SETCV-CONFIG	Set control vector (configuration)
R*SETCV-MAINT	Set control vector (maintenance)
R*STARTMEAS	Start measurement interval
R*STOPMEAS	Stop measurement interval
R*TERM-OTHER	Terminate other
R*TERM-OTHER-CD	Terminate other cross-domain
R*TERM-SELF	Terminate self format 0
R*TERM-SELF_F1	Terminate self format 1
R*TESTMODE	Test mode
R*UNBINDF	Unbind failure
R*VR-INOP	Virtual route inoperative

Table 13–4 FMD Request/Response Unit Identifiers [continued]

Table 13-5 shows defined RU (request/response unit) ITL constants for a DFC (data flow control) RU category.

ITL Constant	Description
R*BID	Bid
R*BIS	Bracket initiation stopped
R*CANCEL	Cancel
R*CHASE	Chase
R*LUSTAT	Logical unit status
R*QC	Quiesce complete
R*QEC	Quiesce at end of chain
R*RELQ	Release quiesce
R*RSHUTD	Request shutdown
R*RTR	Ready to receive
R*SBI	Stop bracket initiaion
R*SHUTC	Shutdown complete
R*SHUTD	Shutdown
R*SIG	Signal

Table 13–5 DFC Request/Response Unit Identifiers

Table 13-6 shows defined RU (request unit) ITL constants for the SC (session control) RU category.

ITL Constant	Description
R*ACTCDRM	Activate cross-domain resource manager
R*ACTLU	Activate logical unit
R*ACTPU	Activate physical unit
R*BIND	Bind session
R*CLEAR	Clear
R*CRV	Cryptography verification
R*DACTCDRM	Deactivate cross-domain resource manager
R*DACTLU	Deactivate logical unit
R*DACTPU	Deactivate physical unit
R*RQR	Request recovery
R*SDT	Start data traffic
R*STSN	Set and test sequence numbers
R*UNBIND	Unbind session

Table 13–6 SC Request/Response Unit Identifiers

Table 13–7 shows defined RU (request unit) ITL constants for the NC (network control) RU category.

ITL Constant	Description
R*ANSC	Contents of NC header = 06
R*LSA	Lost subarea
R*NC–ACTVR	NC activate virtual route
R*NC–DACTVR	NC deactivate virtual route
R*NC–ER–ACT	NC explicit route activate
R*NC–ER–ACT–REPLY	NC explicit route activate reply
R*NC–ER–INOP	NC explicit route inoperative
R*NC–ER–OP	NC explicit route operative
R*NC–ER–TEST	NC explicit route test
R*NC–ER–TEST–REPLY	NC explicit route test reply
R*NC–IPL–ABORT	NC IPL abort
R*NC–IPL–FINAL	NC IPL final
R*NC–IPL–INIT	NC IPL initial
R*NC–IPL–TEXT	NC IPL text

Table 13–7 NC Request/Response Unit Identifiers

?MATCH_I (string-- flag)
Returns true if a user-defined character string is found.

This is an *anchored* match, i.e. a byte-for-byte match starting at the first byte of the information field of the received I frame.

NOTE
To accommodate "don't care" character positions, the question mark character for ASCII, or hex 3F character can be used.

Example:
Match an ACTLU response unit within an FID2 transmission header.

$$X'\ \underbrace{3F3F3F3F3F3F}_{\text{FID2}}\underbrace{EB3F3F}_{\text{RH}}0D'\ \ ?MATCH\_I$$

The first six "don't care" characters include an FID2 transmission header. The hex 'EB' in byte 0 of the RH indicates that this frame contains a response header with a session control RU. The next two "don't care" characters include bytes 1 and 2 of the RH. The hex '0D' is the code for an ACTLU response unit.

Example:
This example shows an alternate method of matching an ACTLU response unit within an FID2. This alternate code is useful for it does not require knowledge of actual bits within a desired frame.

```
FID-ID @ R*FID2 =            ( Is this an FID2? )
?RH_RES  AND                 ( Is this also a Response? )
R*SC ?RX_RH  AND             ( Is this also a Session Control RU? )
R*ACTLU ?RX_RU  AND          ( Is this also an ACTLU? )
```

?SEARCH_I (string-- flag)
Returns true if a user-defined character string is found within the information field of an I frame.

This is an *unanchored* match, i.e. searches for an exact match anywhere in the received I frame regardless of the position.

Example:
Search for an ACTLU RU within a frame.

```
X" OD" ?SEARCH_I
```

This code returns a true flag if an ACTLU frame is received. However, it also returns a true flag for any I frame with the value hex 0D at any location within the information field.

Timeout Detection

There are 128 user programmable timers available. Timers 1 through 24 can be used in the test manager. Timer 32 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 F1." TCR
         1 NEW_STATE
     }ACTION
  }STATE
```

TIMER–NUMBER (-- address)
Contains the number of the expired timer. Valid values are 1 through 128.

Function Key Detection

Refer to the Programmer's Reference Manual.

Interprocessor Mail Events

Refer to the Programmer's Reference Manual.

Wildcard Events

The SNA Monitor and SDLC Emulation support 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: EOF_IND, TIMER_IND, I/F_IND, FK_IND or COMMAND_IND.

EOF_IND (-- value)
A constant value in the EVENT–TYPE variable when the received event is a frame. The actual protocol type is in the FRAME–ID, FID–ID and RH–ID variables.

TIMER_IND (-- value)
A constant value in the EVENT–TYPE variable when the received frame is a timeout.

I/F_IND (-- 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.

FK_IND (-- value)
A constant value in the EVENT–TYPE variable when a function or cursor key is detected.

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 SDLC/SNA Actions

All of the general actions explained in the Programmer's Reference Manual are supported in the SNA Monitor and SDLC Emulation. For additional display commands specific to SDLC/SNA, 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 13–8 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 13–9 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 13–10 V.36/RS–449 Interface Lead Transitions

Protocol Actions

Frames can be transmitted using any of the commands described in Section 12.

NOTE
Any I frames or UI frames to be transmitted by the emulation are queued up and sent when possible. Once a UI frame is transmitted, it is removed from the queue and unavailable for retransmission. When an I frame is sent, it is maintained in a backup queue until it is acknowledged. Each data link maintains its own separate queue.

RESET_QUEUES (queue id --)
Resets the specified queue and discards any queued frames.

Queue ID	Description
UI_QUEUE	UI queue
I_QUEUE	I queue
I_BACKUP	I frame backup queue (the backup queue contains frames for retransmission)

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 to any size desired. However, the SDLC Emulation limits the number of bytes that can be transmitted to 4096.

A buffer consists of four bytes with values of 0, two bytes containing the length of the frame, and the remaining bytes consisting of user-defined contents.

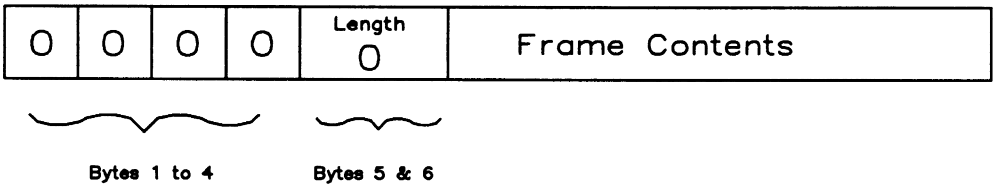


Figure 13–1 Buffer Structure

NOTE
All buffers are cleared when the TCLR command is issued. TCLR is usually the first command executed 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 sent is defined by the frame size set on the Emulation Configuration Menu or by the value stored in the MAX-FRAME-LENGTH variable.

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.

SEND_BUFFER (buffer number --)

Transmits the specified buffer as the information field carried within an information frame that utilizes the current secondary address and N(R) and N(S) values.

Example:

Create the text in the buffer, then transmit the buffer.

```
" IDACOM" 4  STRING->BUFFER              ( Create text )
4 SEND_BUFFER                            ( 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 SEND_BUFFER                            ( 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 BAR_GRAPH.T

The BAR_GRAPH.T script illustrates the structure and usage of a test script in the test manager to collect statistics and display them in a bar chart fashion. Test script function keys are defined to display statistics or data.

```
TCLR                                ( Clear Test Manager )

#IFNOTDEF  rr_count                 ( Check to see if variables already exist )

      0 VARIABLE  rr_count          ( RR frame counter )
      0 VARIABLE  i_count           ( I frame counter )
      0 VARIABLE  disc_count        ( DISC frame counter )
      0 VARIABLE  snrm_count        ( SNRM frame counter )
      0 VARIABLE  ua_count          ( UA frame counter )
      45 VARIABLE  time             ( Make timeout value 45 tenths of seconds )
      0 VARIABLE  time_spot         ( Starting column position for time printout )

#ENDIF

0 SEQ{  ( -- )                      ( Sequence to reset variables and update time )
      0  rr_count !
      0  i_count !
      0  snrm_count !
      0  disc_count !
      0  ua_count !
      10  time @ START_TIMER               ( Restart the interval timer )
      2  time_spot  @  THERE               ( Position cursor for time printout )
      time  @  10  /MOD  2  W.R            ( Print out the seconds )
      W." ." W.                            ( Print out the tenths of a second )
  }SEQ
```

```
1 SEQ{  ( amount \ color -- )         ( Sequence to display a bar-graph )
     WHERE  DROP  9  THERE            ( Position cursor for bar graph )
     OVER  4  W.R                     ( Print out the amount )
     "  |"  COUNT  CYA_FG  W.TYPE_A
     OVER  50  >                      ( Is it too big for the graph? )
     IF
         2DROP  50  RED_BG            ( Yes so limit it and make graph red )
     ENDIF
     OVER  >R                         ( Save a copy of the length )
     "                                                   "( Bar graph-52 spaces )
     1+ ROT  ROT  W.TYPE_A
                                      ( Output the right number of dots )
     "                                                   "( Unbar graph-52 spaces )
     1+ 50  R>  -  0  W.TYPE_A
                                      ( Blank the rest of the bar graph )
  }SEQ

0 STATE_INIT{                         ( Initialize the user window )
     " SHOW DATA"  1  LABEL_KEY       ( Put up function keys )
     " SHOW GRAPH"  2  LABEL_KEY
     3  CLEAR_KEY  4  CLEAR_KEY  5  CLEAR_KEY
     6  CLEAR_KEY  7  CLEAR_KEY  8  CLEAR_KEY
     OPEN_USER  CLEAR_TEXT  0  PAINT  SHOW_USER    ( Clear screen )
     1  30  THERE                                  ( Position for title )
     " SDLC Line Statistics"  COUNT  WHI_FG  W.TYPE_A
     2  27  THERE                                  ( Position for interval time )
     " Frames per Interval  Interval =     "  COUNT  CYA_FG  W.TYPE_A
     WHERE  4  -  time_spot  !  DROP               ( Remember time position )
     " Seconds"  COUNT  CYA_FG  W.TYPE_A
     3  14  THERE
     " 0____+____10___+____20___+____30___+____40___+____50"
     COUNT  CYA_FG  W.TYPE_A                       ( Graph axis )
     10  1  START_TIMER  CLOSE_WINDOW
  }STATE_INIT

0 STATE{
     R*RR  ?RX_BLU
     ACTION{
         1  rr_count  +!              ( Increment the RR count )
     }ACTION

     R*I  ?RX_BLU
     ACTION{
         1  i_count  +!               ( Increment the I-FRAME count )
     }ACTION

     R*DISC  ?RX_BLU
     ACTION{
         1  disc_count  +!            ( Increment the DISC count )
     }ACTION
```

```
    R*SNRM  ?RX_BLU
    ACTION{
        1  snrm_count  +!              ( Increment the SNRM count )
    }ACTION

    R*UA  ?RX_BLU
    ACTION{
        1  ua_count  +!                ( Increment the UA count )
    }ACTION
    ?TIMEOUT
    ACTION{
        OPEN_USER
        4  0  THERE  W." RR's" rr_count  @  GRN_BG  1  RUN_SEQ
        5  0  THERE  W." I FRAME's"  i_count  @  CYA_BG  1  RUN_SEQ
        6  0  THERE  W." DISC's"  disc_count  @  YEL_BG  1  RUN_SEQ
        7  0  THERE  W." SNRM's"  snrm_count  @  MAG_BG  1  RUN_SEQ
        8  0  THERE  W." UA's"  ua_count  @  WHI_BG  1  RUN_SEQ
        0  RUN_SEQ  CLOSE_WINDOW
    }ACTION
}STATE
```

14.2 SNA_STAT.F

The SNA_STAT.F script illustrates the structure and usage of a test script in the SNA Monitor which collects and displays frame and transmission header statistics. This scenario can be used when monitoring line data or when playing back from disk recordings or capture RAM. Test script function keys are defined to display statistics, data or clear the statistic counts.

```
( -------------------------------------------------------------)
( SNA_STAT.F  : SNA  ANALYZER  Test Manager Scenario            )
(               Collect and display SNA statistics              )
(               COUNTER1 to COUNTER30 collect counts.           )
(       f1 = Display Statistics.                                )
(       f2 = Display Data.                                      )
(       f3 = Clear Statistic counts.                            )
( -------------------------------------------------------------)

#IFNOTDEF  ZERO_CNT      ( Use conditional compiling to create counter    )
                         ( Variables and ZERO_CNT word in user dictionary )
                         ( only if they have not been entered before.     )

0 VARIABLE COUNTER11    0 VARIABLE COUNTER12    0 VARIABLE COUNTER13
0 VARIABLE COUNTER14    0 VARIABLE COUNTER15    0 VARIABLE COUNTER16
0 VARIABLE COUNTER17    0 VARIABLE COUNTER18    0 VARIABLE COUNTER19
0 VARIABLE COUNTER20    0 VARIABLE COUNTER21    0 VARIABLE COUNTER22
0 VARIABLE COUNTER23    0 VARIABLE COUNTER24    0 VARIABLE COUNTER25
0 VARIABLE COUNTER26    0 VARIABLE COUNTER27    0 VARIABLE COUNTER28
0 VARIABLE COUNTER29    0 VARIABLE COUNTER30

:  ZERO_CNT  ( -- )      ( Zero statistic counts )
      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 COUNTER20 !
      0 COUNTER21 ! 0 COUNTER22 ! 0 COUNTER23 ! 0 COUNTER24 ! 0 COUNTER25 !
      0 COUNTER26 ! 0 COUNTER27 ! 0 COUNTER28 ! 0 COUNTER29 ! 0 COUNTER30 !
;

#ENDIF                   ( End of the conditional compiling )


TCLR                ( Clear test manager dictionary )

WAKEUP_ON           ( Wakeup the test manager to display statistics on startup )

0 STATE_INIT{
      " STATISTICS"  1  LABEL_KEY
      " SHOW DATA"  2  LABEL_KEY
      " ZERO STAT"  3  LABEL_KEY
      4 CLEAR_KEY   5 CLEAR_KEY   6 CLEAR_KEY   7 CLEAR_KEY   8 CLEAR_KEY
  }STATE_INIT
```

```
3 SEQ{                                  ( Sequence 3 will display statistic screen )
     POP_USER                           ( Open user Display Window )
     CLEAR_TEXT   WHI_FG PAINT          ( Clear screen text and color )
          1 24  THERE  W." SDLC/SNA  Monitor Statistics"
          3 10  THERE  W." SNRM = "       COUNTER1  @ W.  ( Display Statistics )
          4 10  THERE  W." UA   = "       COUNTER2  @ W.
          5 10  THERE  W." DISC = "       COUNTER3  @ W.
          6 10  THERE  W." RR   = "       COUNTER5  @ W.
          7 10  THERE  W." RNR  = "       COUNTER6  @ W.
          8 10  THERE  W." REJ  = "       COUNTER7  @ W.
          9 10  THERE  W." DM   = "       COUNTER8  @ W.
         10 10  THERE  W." FRMR = "       COUNTER9  @ W.
         11 10  THERE  W." SIM  = "       COUNTER10 @ W.
          3 30  THERE  W." RIM       = "  COUNTER11 @ W.
          4 30  THERE  W." UI        = "  COUNTER12 @ W.
          5 30  THERE  W." UP        = "  COUNTER13 @ W.
          6 30  THERE  W." RD        = "  COUNTER14 @ W.
          7 30  THERE  W." XID       = "  COUNTER15 @ W.
          8 30  THERE  W." TEST      = "  COUNTER16 @ W.
         10 30  THERE  W." I FRAME   = "  COUNTER4  @ W.
         11 30  THERE  W." BAD FRAME = "  COUNTER25 @ W.
          3 50  THERE  W." FID0 = "       COUNTER18 @ W.
          4 50  THERE  W." FID1 = "       COUNTER19 @ W.
          5 50  THERE  W." FID2 = "       COUNTER20 @ W.
          6 50  THERE  W." FID3 = "       COUNTER21 @ W.
          7 50  THERE  W." FID4 = "       COUNTER22 @ W.
          8 50  THERE  W." FIDF = "       COUNTER23 @ W.
          9 50  THERE  W." INVF = "       COUNTER24 @ W.
     CLOSE_WINDOW
  }SEQ

4 SEQ{              ( Define sequence 4 to update and display I frame count )
     1  COUNTER4  +!                    ( Statistic updating for I frames )
     10  40  THERE
     COUNTER4  @  W.                    ( Display I frame count )
  }SEQ
```

```
5 SEQ{
    FID-ID  @                    ( Decode transmission header type )
    DOCASE
        CASE R*FID0
        { OPEN_USER 1 COUNTER18 +! 3 57 THERE COUNTER18 @ W. CLOSE_WINDOW }
        CASE R*FID1
        { OPEN_USER 1 COUNTER19 +! 4 57 THERE COUNTER19 @ W. CLOSE_WINDOW }
        CASE R*FID2
        { OPEN_USER 1 COUNTER20 +! 5 57 THERE COUNTER20 @ W. CLOSE_WINDOW }
        CASE R*FID3
        { OPEN_USER 1 COUNTER21 +! 6 57 THERE COUNTER21 @ W. CLOSE_WINDOW }
        CASE R*FID4
        { OPEN_USER 1 COUNTER22 +! 7 57 THERE COUNTER22 @ W. CLOSE_WINDOW }
        CASE R*FIDF
        { OPEN_USER 1 COUNTER23 +! 8 57 THERE COUNTER23 @ W. CLOSE_WINDOW }
        CASE R*INVFID
        { OPEN_USER 1 COUNTER24 +! 9 57 THERE COUNTER24 @ W. CLOSE_WINDOW }
    ENDCASE
  }SEQ

0 STATE{           ( Define state 0 to collect statistics and check events )
    R*SNRM  ?RX_BLU                   ( SNRM received )
    ACTION{
        OPEN_USER                     ( Open user Display Window )
        1  COUNTER1  +!               ( Update SNRM counter )
        3  17 THERE  COUNTER1 @  W.   ( Display SNRM count )
        CLOSE_WINDOW                  ( Close Display Window )
    }ACTION

    R*UA  ?RX_BLU                     ( UA received )
    ACTION{
        OPEN_USER                     ( Open user Display Window )
        1  COUNTER2  +!               ( Update UA counter )
        4  17 THERE  COUNTER2 @  W.   ( Display UA count )
        CLOSE_WINDOW                  ( Close Display Window )
    }ACTION

    R*DISC  ?RX_BLU                   ( DISC received )
    ACTION{
        OPEN_USER                     ( Open user Display Window )
        1  COUNTER3  +!               ( Update DISC counter )
        5  17 THERE  COUNTER3 @  W.   ( Display DISC count )
        CLOSE_WINDOW                  ( Close Display Window )
    }ACTION

    R*RR  ?RX_BLU                     ( RR frame received )
    ACTION{
        OPEN_USER                     ( Open user Display Window )
        1  COUNTER5  +!               ( Update RR frame counter )
        6  17 THERE  COUNTER5 @  W.   ( Display RR frame count )
        CLOSE_WINDOW                  ( Close Display Window )
    }ACTION
```

```
R*RNR?RX_BLU                        ( RNR frame received )
ACTION{
    OPEN_USER                       ( Open user Display Window )
    1  COUNTER6  +!                 ( Update RNR frame counter )
    7  17 THERE  COUNTER6 @  W.     ( Display RNR frame count )
    CLOSE_WINDOW                    ( Close Display Window )
}ACTION

R*REJ  ?RX_BLU                      ( REJ frame received )
ACTION{
    OPEN_USER                       ( Open user Display Window )
    1 COUNTER7 +!                   ( Update REJ frame counter )
    8 17 THERE  COUNTER7 @  W.      ( Display REJ frame count )
    CLOSE_WINDOW                    ( Close Display Window )
}ACTION

R*DM ?RX_BLU                        ( DM frame received )
ACTION{
    OPEN_USER                       ( Open user Display Window )
    1 COUNTER8 +!                   ( Update DM frame counter )
    9 17 THERE  COUNTER8 @  W.      ( Display DM frame count )
    CLOSE_WINDOW                    ( Close Display Window )
}ACTION

R*FRMR ?RX_BLU                      ( FRMR frame received )
ACTION{
    OPEN_USER                       ( Open user Display Window )
    1 COUNTER9 +!                   ( Update FRMR frame counter )
    10 17 THERE  COUNTER9 @  W.     ( Display FRMR frame count )
    CLOSE_WINDOW                    ( Close Display Window )
}ACTION

R*SIM ?RX_BLU                       ( Set Initialization Mode )
ACTION{
    OPEN_USER                       ( Open user Display Window )
    1 COUNTER10 +!                  ( Update SIM frame count )
    11 17 THERE  COUNTER10 @  W.    ( Display SIM frame count )
    CLOSE_WINDOW                    ( Close Display Window )
}ACTION

R*RIM ?RX_BLU                       ( Request Initialization Mode )
ACTION{
    OPEN_USER                       ( Open user Display Window )
    1 COUNTER11 +!                  ( Update RIM counter )
    3 42 THERE  COUNTER11 @  W.     ( Display RIM count )
    CLOSE_WINDOW                    ( Close Display Window )
}ACTION
```

```
R*UI ?RX_BLU                         ( Unnumbered Information Frame )
ACTION{
    OPEN_USER                        ( Open user Display Window )
    1 COUNTER12 +!                   ( Update UI counter )
    4 42 THERE  COUNTER12 @  W.      ( Display UI count )
    CLOSE_WINDOW                     ( Close Display Window )
}ACTION

R*UP ?RX_BLU                         ( Unnumbered Poll )
ACTION{
    OPEN_USER                        ( Open user Display Window )
    1 COUNTER13 +!                   ( Update UP counter )
    5 42 THERE  COUNTER13 @  W.      ( Display UP count )
    CLOSE_WINDOW                     ( Close Display Window )
}ACTION

R*RD  ?RX_BLU                        ( Request Disconnect received )
ACTION{
    OPEN_USER                        ( Open user Display Window )
    1 COUNTER14 +!                   ( Update RD counter )
    6 42 THERE  COUNTER14 @  W.      ( Display RD count )
    CLOSE_WINDOW                     ( Close Display Window )
}ACTION

R*XID ?RX_BLU                        ( X I D frame )
ACTION{
    OPEN_USER                        ( Open user Display Window )
    1 COUNTER15 +!                   ( Update XID count )
    7 42 THERE  COUNTER15 @  W.      ( Display XID count )
    CLOSE_WINDOW                     ( Close Display Window )
}ACTION

R*TEST ?RX_BLU                       ( TEST Frame )
ACTION{
    OPEN_USER                        ( Open user Display Window )
    1 COUNTER16 +!                   ( Update Test frame count )
    8 42 THERE  COUNTER16 @  W.      ( Display Test frame count )
    CLOSE_WINDOW                     ( Close Display Window )
}ACTION

R*I ?RX_BLU                          ( Information frame )
ACTION{
    OPEN_USER                        ( Open user Display Window )
    1 COUNTER4 +!                    ( Update I frame count )
    10 42 THERE  COUNTER4 @  W.      ( Display I frame count )
    CLOSE_WINDOW                     ( Close display window )
    5 RUN_SEQ                        ( Decode transmission header )
}ACTION
```

```
    R*INVFRM ?RX_BLU                ( Invalid frame received )
    ACTION{
        OPEN_USER                   ( Open user Display Window )
        1 COUNTER25  +!             ( Update invalid frame count )
        11 42 THERE  COUNTER25 @  W.  ( Display invalid frame count )
        CLOSE_WINDOW                ( Close Display Window )
    }ACTION

    UF3 ?KEY   ?WAKEUP OR           ( CTRL f1 key or TM_RUN wakeup events )
    ACTION{
        ZERO_CNT 3 RUN_SEQ          ( Open and show user Display Window )
    }ACTION

    UF2 ?KEY
    ACTION{
        SHOW_DATA                   ( Show Data Window )
    }ACTION

    UF1 ?KEY
    ACTION{
        3 RUN_SEQ                   ( Show user Display Window )
    }ACTION
}STATE                              ( END STATE 0 )
```

14.3 SNA 3274 Scenario

The file 3274.F can be used to test a 3274 terminal. The file 3274_DTE.F is provided to simulate a 3274 terminal.

This test script emulates a 37x5 communications controller generating traffic to a 327x and 3178 type CRT.

Pressing the *START* function key initiates the emulation and:

- a link is established;
- an ACTPU request is transmitted;
- when an ACTPU response is received, a 'Welcome' message is transmitted to the 3178;
- when either an INIT-SELF (format 0) request is received or a 'log-on' is entered on the 3178 keyboard, a BIND request is transmitted;
- when a BIND response is received, the function keys are relabelled. The user has the option of generating single or continuous screens of data which are transmitted to the 3178 CRT;
- entries typed on the emulation partner's keyboard receive a response which unlocks the keyboard; and
- when a 'logoff' message is typed on the 3178 keyboard, the tester initiates the UNBIND, DACTLU, DACTPU and DISC exchanges.

This test script has been designed to run on a switched SDLC line. To run on a non-switched SDLC line, replace the command '0 ID-OUT !' with one which specifies the station ID.

Example:
Set the link address to 41.

```
41 1 DO_SETID
File 3274.F

TCLR

BLU_BG TCOLOR                   ( Set trace statement color )

RESET                           ( Reset parameters and terminate link )
0 ID-OUT !                      ( Set PU address to zero )
3 SDAF !                        ( Set destination address )
1 SOAF !                        ( Set origin address )
RFRM-INV RFRM-RR                ( Block RR and invalid frames from display )
7 1 DO_SETW                     ( Set window size to 7 )

CTS_ON                          ( Turn on leads )
DSR_ON
CD_ON
SQ_ON

" SEG_2" 0 FILE->BUFFER         ( Create buffer )
" SEG_3" 1 FILE->BUFFER
" SEG_4" 2 FILE->BUFFER
" SEG_5" 3 FILE->BUFFER
" LAST_SET" 4 FILE->BUFFER
```

```
#IFNOTDEF S:ACTPU      ( Compile following code only if not in user dictionary )

: S:ACTPU ( -- )                             ( Send Activate Physical Unit RU )
    X" 2D00000004D36B80001102010500000000001"        ( Create string )
    SEND_BTU                                 ( Send frame )
;

: S:ACTLU ( -- )                             ( Send Activate Logical Unit RU )
    X" 2D00000004D46B80000D0201"             ( Create string )
    SDAF @ SET_DAF                           ( Set destination address )
    SEND_BTU                                 ( Send frame )
;

: S:INIT-SELF_F0_RES ( -- )                  ( Send Initiate-Self Format 0 RU )
    X" 2C00000000008B8000010681"             ( Create string )
    RETURN_FIELDS                            ( Echo back received DAF, OAF, and SNF )
    SEND_BTU                                 ( Send frame )
;

: S:BIND ( -- )                              ( Send Bind Session RU )
    X" 2D00020104D56B800031010303B1903080000087F80000020000000000185
    000007E000003E3E2D600"                   ( Create string )
    SDAF @ SET_DAF                           ( Set destination address )
    SOAF @ SET_OAF                           ( Set origin address )
    SEND_BTU                                 ( Send frame )
;

: S:SDT ( -- )                               ( Send Start Data Traffic RU )
    X" 2D00020104D66B8000A0"                 ( Create string )
    SDAF @ SET_DAF                           ( Set destination address )
    SOAF @ SET_OAF                           ( Set origin address )
    SEND_BTU                                 ( Send frame )
;

: S:UNBIND ( -- )                            ( Send Unbind Session RU )
    X" 2D00020104D76B80003202"               ( Create string )
    SDAF @ SET_DAF                           ( Set destination address )
    SOAF @ SET_OAF                           ( Set origin address )
    SEND_BTU                                 ( Send frame )
;

: S:DACTLU ( -- )                            ( Send Deactivate Logical Unit RU )
    X" 2D00020004D86B80000E"                 ( Create string )
    SDAF @ SET_DAF                           ( Set destination address )
    SEND_BTU                                 ( Send frame )
;

: S:DACTPU ( -- )                            ( Send Deactivate Physical Unit RU )
    X" 2D00000004D96B80001202"               ( Create string )
    SEND_BTU                                 ( Send frame )
;
```

```
: S:FMD_ACK ( -- )                        ( Send Response Header Acknowledgment )
    X" 2C0000000000838000"                ( Create string )
    RETURN_FIELDS                         ( Echo back received DAF, OAF, and SNF )
    SEND_BTU                              ( Send frame )
;

: S:DATA_RES ( -- )                       ( Send data )
    X" 2C000301000A038040F5C111C2601DC8D9C5E2D7D6D5E2C540E3D640C5D5E3D9E8"
                                          ( Create string )
    SDAF @ SET_DAF                        ( Set destination address )
    SOAF @ SET_OAF                        ( Set origin address )
    SSNF @ SET_SNF                        ( Set sequence number )
    1 SSNF +!                             ( Increment sequence number )
    SEND_BTU                              ( Send frame )
;

: S:LOGON_RES                             ( Send logon message )
    X" 2C000301000A0380C0F5C111C2601DC8D3D6C7D6D540C9E240E2E4C3C3C5E2E2C6E4D31D4011
    C3F013"                               ( Create string )
    SDAF @ SET_DAF                        ( Set destination address )
    SOAF @ SET_OAF                        ( Set origin address )
    SSNF @ SET_SNF                        ( Set sequence number )
    1 SSNF +!                             ( Increment sequence number )
    SEND_BTU                              ( Send frame )
;

: S:LOGOFF_RES                            ( Send logoff message )
    X" 2C000301000A038040F5C111C2601DC8D3D6C7D6C6C640C9E240E2E4C3C3C5E2E2C6E4D31D40
    11C3F013"                             ( Create string )
    SDAF @ SET_DAF                        ( Set destination address )
    SOAF @ SET_OAF                        ( Set origin address )
    SSNF @ SET_SNF                        ( Set sequence number )
    1 SSNF +!                             ( Increment sequence number )
    SEND_BTU                              ( Send frame )
;
```

```
: SEND_SCREEN ( -- )
    " xxxxxxxxxxxxxxx\\\@@@@@C "@@#"#@"K@@'''@b@$@p@'''@@@@@\\\APqqqqqqqqqqqqqqqqqq
qqqq@@@@@s @q@@@@@qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqB'rrrrrrrrrrrrrrrrrrrrr@
@@@s @r@@@@"                          ( Create string )
    DUP 1+
    X" 2800030100020380C0F1C31140C1"    ( Create transmission header )
    COUNT >R SWAP R> CMOVE              ( Move into string )
    DUP 62 + DUP C@ 1+ DUP              ( Get screen number )
    0XFA =                              ( Is it ten ? )
    IF                                  ( Yes )
        DROP 0XF0                       ( Set it back to zero )
    ENDIF
    SWAP C!
    SOAF @ SET_OAF                      ( Set origin address )
    SDAF @ SET_DAF                      ( Set destination address )
    SSNF @ SET_SNF                      ( Set sequence number )
    SEND_BTU                            ( Send frame )
    0 BUFFER 5 + SOAF @ SET_OAF SDAF @ SET_DAF SSNF @ SET_SNF DROP 0 SEND_BUFFER
    1 BUFFER 5 + SOAF @ SET_OAF SDAF @ SET_DAF SSNF @ SET_SNF DROP 1 SEND_BUFFER
    2 BUFFER 5 + SOAF @ SET_OAF SDAF @ SET_DAF SSNF @ SET_SNF DROP 2 SEND_BUFFER
    3 BUFFER 5 + SOAF @ SET_OAF SDAF @ SET_DAF SSNF @ SET_SNF DROP 3 SEND_BUFFER
    4 BUFFER 5 + SOAF @ SET_OAF SDAF @ SET_DAF SSNF @ SET_SNF DROP 4 SEND_BUFFER
    ( Set up destination address, origin address and sequence number in buffers -
    send buffers )
    1 SSNF +!                           ( Increment sequence number )
;

#ENDIF

0 STATE_INIT{
      RESET                             ( Reset emulation states and variables )
      0 COUNTER1  !                     ( Clear logon status )
      " START" 1 LABEL_KEY              ( Label keys )
      2 CLEAR_KEY    3 CLEAR_KEY    4 CLEAR_KEY    5 CLEAR_KEY
      6 CLEAR_KEY    7 CLEAR_KEY    8 CLEAR_KEY
      " Enter f1 under TestKeys Topic to start test"  W.NOTICE
  }STATE_INIT

0 STATE{                                ( Establish SDLC link )
      UF1 ?KEY
      ACTION{
          RESET                         ( Reset variables and terminate link )
          "  " 1 LABEL_KEY              ( Clear label from key )
          0 ID-OUT !                    ( Set PU address to zero )
          1 SSNF !                      ( Set sequence number to one )
          FORM_LINK                     ( Establish the link )
          1 NEW_STATE
      }ACTION
  }STATE
```

```
1 STATE{                                ( Activate the PU )
      R*UA ?RX_BLU
      ACTION{
          S:ACTPU 2 NEW_STATE           ( Send ACTPU )
      }ACTION
  }STATE

2 STATE{                                ( Activate the LU )
      ?RH_RES R*ACTPU ?RX_RU AND        ( ACTPU response? )
      ACTION{
          S:ACTLU
          3 NEW_STATE
      }ACTION
  }STATE

3 STATE{
      ?RH_RES R*ACTLU ?RX_RU AND        ( ACTLU response? )
      ACTION{
          " , fESCVTE@cV@cHE@IDACVT@Wc@srwt@DYIeEY@WYVGYAT"
                                        ( Create welcome message )
          SDAF @ SET_DAF                ( Set destination address
          SEND_BTU                      ( Send WELCOME message )
          " WAITING FOR INIT-SELF OR 'logon'" W.NOTICE
          4 NEW_STATE
      }ACTION
  }STATE

4 STATE{                                ( Wait for INIT-SELF )
      ?RH_REQ R*INIT-SELF_F0 ?RX_RU AND
      ACTION{
          S:INIT-SELF_F0_RES            ( Send INIT-SELF response )
          S:BIND                        ( Send BIND request )
          5 NEW_STATE
      }ACTION

      R*FMD ?RX_RH ?RH_REQ AND " " ?SEARCH_I AND
      ACTION{
          S:BIND                        ( Send BIND request )
          1 COUNTER1 !                  ( Status to indicate logon received )
          5 NEW_STATE
      }ACTION
  }STATE

5 STATE{                                ( Wait for a BIND response )
      ?RH_RES R*BIND ?RX_RU AND         ( BIND response? )
      ACTION{
          S:SDT                         ( Send SDT )
          6 NEW_STATE
      }ACTION
  }STATE
```

```
6 STATE{                                    ( Wait for SDT response )
     ?RH_RES R*SDT ?RX_RU AND
     ACTION{
          1 SSNF !
          COUNTER1 @                        ( Find out if logon received )
          IF
               S:LOGON_RES
          ENDIF
          " Enter F2 to send Data once, F3 to send continuously" W.NOTICE
          1 CLEAR_KEY
          " ONE_SCN"  2 LABEL_KEY
          " CONT_SCN" 3 LABEL_KEY
          7 NEW_STATE
     }ACTION
  }STATE

7 STATE{
     UF2 ?KEY
     ACTION{                                ( Send DATA )
          SEND_SCREEN
     }ACTION

     UF3 ?KEY
     ACTION{
          SEND_SCREEN                       ( Send one screen of data )
          "  "  DUP 2 LABEL_KEY  3 LABEL_KEY
          8 NEW_STATE
     }ACTION

     R*FMD ?RX_RH ?RH_REQ AND " " ?SEARCH_I AND
     ACTION{                                ( Send UNBIND request )
          S:LOGOFF_RES
          S:UNBIND
          9 NEW_STATE
     }ACTION

     R*I ?RX_BLU ?RH_REQ AND R*WHOLE-BIU ?MAPPING-FIELD
     R*LAST-SEGMENT ?MAPPING-FIELD OR AND
                                            ( Does the RH contain an entire message or
                                            last segment of message? )
     ACTION{
          S:FMD_ACK                         ( Send acknowledgment )
          CHANGE-DIR-IND @                  ( Was the change indicator set? )
          IF                                ( Yes )
               S:DATA_RES                   ( Send data )
          ENDIF
     }ACTION
  }STATE
```

```
8 STATE[
     R*FMD ?RX_RH ?RH_RES AND SENSE-DATA-IND @ R*NONE = AND
                                             ( Is this an FMD response header without
                                             sense data? )
     ACTION[
         SEND_SCREEN
     ]ACTION

     R*FMD ?RX_RH ?RH_REQ AND " " ?SEARCH_I AND
                                             ( Is this an FMD request header containing
                                             this string? )
     ACTION[                                 ( Send UNBIND request )
         S:LOGOFF_RES                        ( Send logoff )
         S:UNBIND                            ( Send UNBIND )
         9 NEW_STATE
         2 CLEAR_KEY    3 CLEAR_KEY
     ]ACTION

     R*I ?RX_BLU ?RH_REQ AND R*WHOLE-BIU ?MAPPING-FIELD
     R*LAST-SEGMENT ?MAPPING-FIELD OR AND
                                             ( Does the RH contain an entire message or
                                             last segment of message? )
     ACTION[
         S:FMD_ACK                           ( Send an acknowledgment )
         CHANGE-DIR-IND @                    ( Is the change indicator set? )
         IF                                  ( Yes )
             S:DATA_RES                      ( Send data )
         ENDIF
     ]ACTION
 ]STATE


9 STATE[                                     ( Wait for UNBIND response )
     ?RH_RES R*UNBIND ?RX_RU AND
     ACTION[                                 ( Send DACTLU )
         S:DACTLU                            ( Send DACTLU request )
         10 NEW_STATE
     ]ACTION
 ]STATE

10 STATE[                                    ( Wait for a DACTLU response )
     ?RH_RES R*DACTLU ?RX_RU AND
     ACTION[
         S:DACTPU                            ( Send DACTPU request )
         11 NEW_STATE
     ]ACTION
 ]STATE
```

```
11 STATE{                          ( Wait for a DACTPU response )
      ?RH_RES R*DACTPU ?RX_RU AND
      ACTION{                      ( Send DISC )
          DISC
          12 NEW_STATE
      }ACTION
  }STATE

12 STATE{                          ( Wait for UA )
      R*UA ?RX_BLU
      ACTION{
          T." Test is completed." TCR   ( Reset parameters and terminate link )
          TM_STOP                  ( Stop test manager )
      }ACTION
  }STATE
```

3274_DTE.F

This test script is developed to simulate a 3274 terminal (for example, as a 327x cluster controller and 3178 type CRT emulation).

```
( FILE TITLE : 3274_DTE.F  )
( DTE Emulation scenario used as test partner with 3274.F Scenario )

TCLR
BLU_BG  TCOLOR

0xC1 1 DO_SETID                         ( 1 Link address active )
1 1 DO_#LINKS                           ( Use C1 as the link address )
1  SDAF !                               ( Set destination address )
3 SOAF  !                               ( Set origin address )

#IFNOTDEF  S:LOGON

: S:LOGON
      X" 2D000000000010380007DC6D311C5C89396879695"      ( Define string )
      SSNF @ SET_SNF                    ( Set sequence number )
      SDAF @ SET_DAF                    ( Set destination address )
      SOAF @ SET_OAF                    ( Set origin address )
      SEND_BTU                          ( Send the frame )
;

: S:LOGOFF
      X" 2D000000000010380007DC6D311C5C8939687968686"   ( Define string )
      SSNF @ SET_SNF                    ( Set sequence number )
      SDAF @ SET_DAF                    ( Set destination address )
      SOAF @ SET_OAF                    ( Set origin address )
      SEND_BTU                          ( Send frame )
;

: S:INIT_SELF
      X" 2C00000300010B80000106810004F4C3F3F2F7F8F2F303E3E2D6000009D5
      E2C1F761C1C4D7F6"
      SDAF @  SET_DAF                   ( Set destination address )
      SOAF @  SET_OAF                   ( Set origin address )
      SEND_BTU                          ( Send frame )
;

#ENDIF

0 STATE_INIT{                           ( Label function keys )
      " LOGON"       1 LABEL_KEY
      " LOGOFF"      2 LABEL_KEY
      " INIT SELF"   3 LABEL_KEY
      4 CLEAR_KEY    5 CLEAR_KEY   6 CLEAR_KEY
      7 CLEAR_KEY    8 CLEAR_KEY
  }STATE_INIT
```

```
0 STATE{
      UF1 ?KEY
      ACTION{
          S:LOGON                         ( Send logon )
      }ACTION

      UF2 ?KEY
      ACTION{
          S:LOGOFF                        ( Send logoff )
      }ACTION

      UF3 ?KEY
      ACTION{
          S:INIT_SELF                     ( Send INIT_SELF )
      }ACTION

      R*ACTPU  ?RX_RU                     ( ACTPU received? )
      ACTION{                             ( Send ACTPU response )
          X" 2D0000000B52EB80001102010500000000015"
          RETURN_FIELDS
          SEND_BTU
      }ACTION

      R*ACTLU  ?RX_RU                     ( ACTLU received? )
      ACTION{                             ( Send ACTLU response )
          X" 2D0002000B53EB80000D0201"
          RETURN_FIELDS
          SEND_BTU
      }ACTION

      R*BIND  ?RX_RU                      ( BIND received? )
      ACTION{                             ( Send BIND response )
          X" 2D0008010001EBA0E031"
          RETURN_FIELDS
          SEND_BTU
      }ACTION

      R*SDT  ?RX_RU                       ( SDT received? )
      ACTION{                             ( Send SDT response )
          X" 2D0008010001EBA060A0"
          RETURN_FIELDS
          SEND_BTU
      }ACTION

      R*UNBIND  ?RX_RU                    ( UNBIND received? )
      ACTION{                             ( Send UNBIND response )
          X" 2D0008010001EBA0E032"
          RETURN_FIELDS
          SEND_BTU
      }ACTION
```

```
    R*DACTLU  ?RX_RU                    ( DACTLU received? )
    ACTION{                             ( Send DACTLU response )
        X" 2D0008010001EBA0E00E"
        RETURN_FIELDS
        SEND_BTU
    }ACTION

    R*DACTPU  ?RX_RU                    ( DACTPU received? )
    ACTION{                             ( Send DACTPU response )
        X" 2D0000000B52EB80001202"
        RETURN_FIELDS
        SEND_BTU
    }ACTION

    R*I  ?RX_BLU  ?RH_REQ AND  R*WHOLE-BIU ?MAPPING-FIELD
    R*LAST-SEGMENT ?MAPPING-FIELD  OR  AND
    ( Request header containing entire message or last segment of message )
    ACTION{                             ( Send response )
        X" 2C0000000000838000"
        RETURN_FIELDS
        SEND_BTU
    }ACTION
}STATE
```

14.4 Multipoint Test Scripts

The scenario files 'PRI_MULTI.F' and 'SEC_MULTI.F' are provided as sample test scripts to demonstrate the multipoint capabilities of the SDLC Emulation.

PRI_MULTI.F should be loaded and run on an IDACOM tester configured as a primary. SEC_MULTI.F should be loaded and run on an IDACOM tester configured as a secondary.

PRI_MULTI.F can also be run on the primary when no test script is running on the secondary. The user can select 'ONE QUEUE' or 'CONT QUES', F1 and F2 respectively. ONE QUEUE causes the primary to queue three information frames on each link which are transmitted as soon as there is an opportunity to do so. CONT QUES causes the primary to continuously queue information frames on all links.

If SEC_MULTI.F is running on the secondary, the secondary continuously queues information frames to be transmitted. Frames are then transmitted whenever there is an opportunity to do so.

The text of the information frames is taken from the 'fortune' program running on the UNIX™ operating system.

PRI_MULTI.F

```
( This script is designed to be run on the primary.  It will use #SECS )
( for the  number of links and set each link id to 1 + the link number.  )
( The purpose of this script is to test the multipoint facilities of the )
( SDLC emulation. )

#IFNOTDEF SET_STRINGS
: SET_STRINGS ( -- )                    ( Fill nine buffers with data )
" 'The dA bore is someone who persists in holding his own views after we have en
lightened him with ours. ( 1 )"
0 STRING->BUFFER

" 'The dOne thing the inventors can't seem to get the bugs out of is fresh paint
. ( 2 )"
1 STRING->BUFFER

" 'The dRemember, even if you win the rat race -- you're still a rat. ( 3 )"
2 STRING->BUFFER

" 'The dThere are four kinds of homicide: felonious, excusable, justifiable, and
 praiseworthy...  -- Ambrose Pierce ( 4 )"
3 STRING->BUFFER

" 'The dYou couldn't even prove the White House staff sane beyond a reasonable d
oubt.  -- Ed Meese, on the Hinckley verdict ( 5 )"
4 STRING->BUFFER
```

```
" 'The dGod made the world in six days, and was arrested on the seventh. ( 6 )"
5 STRING->BUFFER

" 'The dTruthful: Dumb and illiterate. ( 7 )"
6 STRING->BUFFER

" 'The dKin: An affliction of the blood ( 8 )"
7 STRING->BUFFER

" 'The dVan Roy's Law: An unbreakable toy is useful for breaking other toys. ( 9
)"
8 STRING->BUFFER

" 'The dGood advice is something a man gives when he is too old to set a bad exa
mple.  -- La Rouchefoucauld ( A )"
9 STRING->BUFFER
;

#ENDIF


TCLR
YES RTRACE                  ( Report, record and capture all trace output )
YES DTRACE
YES CTRACE

0 SEQ{
      #SECS @ 0                          ( Execute actions on each supported link )
      DO
          I LINK# !                      ( Set current link )
          I_BACKUP QEMPTY?  I_QUEUE QEMPTY?
          AND                            ( Are all the I-frames sent? )
          IF
              T" Primary queuing on link " LINK# @ T. TCR
              3 0
              DO
                  J SEND_BUFFER          ( Send 3 more )
              LOOP
          ENDIF
      LOOP
  }SEQ

1 SEQ{                                   ( Execute action on each supported link )
      #SECS @ 0                          ( Reset all links.  This is a quick way to
                                         stop transmissions )
      DO
          I LINK# !                      ( Set current link )
          RESET                          ( Reset )
      LOOP
  }SEQ
```

```
0 STATE_INIT{              ( Done on entry to state 0, sets initial conditions )
      CYA_FG TCOLOR
      MULTI_PT                            ( Choose multipoint configuration )
      SET_STRINGS                         ( Define strings )
      REP_COMP R=ASCII                    ( set up the monitor )
      RH_OFF TH_OFF RU_OFF DATA_CHAR      ( Turn off RH, TH and RU report )
      #SECS @ 10 >
      IF
          10 #SECS !                      ( Only have buffer space to queue
                                          continuously on 10 links )
      ENDIF
      #SECS @ 0
      DO
          I I 1 + SET_LINK_ID             ( Set id's of all links )
      LOOP
      " ONE QUEUE" 1   LABEL_KEY          ( Label function keys )
      " CONT_QUES" 2   LABEL_KEY
      " STOP_ALL " 3   LABEL_KEY
      4 CLEAR_KEY  5 CLEAR_KEY  6 CLEAR_KEY  7 CLEAR_KEY  8 CLEAR_KEY
  }STATE_INIT

0 STATE{
      UF1 ?KEY
      ACTION{
          0 RUN_SEQ                       ( Queue once on each link )
      }ACTION

      UF2 ?KEY
      ACTION{
          0 RUN_SEQ                       ( Queue continuously )
          1 NEW_STATE
      }ACTION

      UF3 ?KEY
      ACTION{
          1 RUN_SEQ                       ( Stop transmissions )
      }ACTION
  }STATE

1 STATE{
      UF1 ?KEY
      ACTION{
          0 NEW_STATE                     ( Go back to one queue )
      }ACTION                             ( Not continuous queuing )

      UF3 ?KEY
      ACTION{
          1 RUN_SEQ                       ( Stop transmission )
          0 NEW_STATE
      }ACTION
```

```
    OTHER_EVENT
    ACTION{
        0 RUN_SEQ                    ( Queue more frames )
    }ACTION
}STATE
```

SEC_MULTI.F

```
( This script is designed to be run on the secondary.  It will use #SECS )
( for the  number of links and set each link id to 1 + the link number.  )
( The purpose of this script is to test the multipoint facilities of the )
( SDLC emulation. )

#IFNOTDEF SET_STRINGS

: SET_STRINGS
" 'The dA bore is someone who persists in holding his own views after we have en
lightened him with ours. ( 1 )"
0 STRING->BUFFER

" 'The dOne thing the inventors can't seem to get the bugs out of is fresh paint
. ( 2 )"
1 STRING->BUFFER

" 'The dRemember, even if you win the rat race -- you're still a rat. ( 3 )"
2 STRING->BUFFER

" 'The dThere are four kinds of homicide: felonious, excusable, justifiable, and
 praiseworthy...  -- Ambrose Pierce ( 4 )"
3 STRING->BUFFER

" 'The dYou couldn't even prove the White House staff sane beyond a reasonable d
oubt.  -- Ed Meese, on the Hinckley verdict ( 5 )"
4 STRING->BUFFER

" 'The dGod made the world in six days, and was arrested on the seventh. ( 6 )"
5 STRING->BUFFER

" 'The dTruthful: Dumb and illiterate. ( 7 )"
6 STRING->BUFFER

" 'The dKin: An affliction of the blood ( 8 )"
7 STRING->BUFFER

" 'The dVan Roy's Law: An unbreakable toy is useful for breaking other toys. ( 9
 )"
8 STRING->BUFFER

" 'The dGood advice is something a man gives when he is too old to set a bad exa
mple.  -- La Rouchefoucauld ( A )"
9 STRING->BUFFER
;

#ENDIF
```

```
TCLR
YES RTRACE                  ( Report, record and capture all trace lines )
YES DTRACE
YES CTRACE

0 STATE_INIT{                              ( Done on entry to state 0 )
      CYA_FG TCOLOR                        ( Sets initial conditions )
      SET_STRINGS
      REP_COMP R=ASCII                     ( Set up the monitor )
      RH_OFF RU_OFF TH_OFF DATA_CHAR       ( Turn off RH, RU, TH reports )
      #SECS @ 10 >
      IF
          10 #SECS !    ( Only buffer space for continuous queuing on 10 links )
      ENDIF
      #SECS @ 0
      DO
          I I 1 + SET_LINK_ID              ( Set the id's of all links )
      LOOP
  }STATE_INIT

0 STATE{
      OTHER_EVENT
      ACTION{
          I_BACKUP QEMPTY?  I_QUEUE QEMPTY?
          AND                              ( Are all the I-frames sent? )
          IF
              T." Secondary queuing frames on link " LINK# @ T. TCR
              3 0
              DO
                  LINK# @ SEND_BUFFER      ( send 3 more I frames )
              LOOP
          ENDIF
      }ACTION
  }STATE
```

14.5 SDLC_SCRIPT.F

This test script illustrates transmission of information frames and the queuing structure of the SDLC Emulation.

Once the script is running, state 0 is entered and the initial conditions in STATE_INIT{ are performed. The link is put into a normal response mode and the monitor is set up to display the frame in character mode. Every time an event occurs, the scenario is run. The scenario continuously runs sequence 0 which checks to see if all the I frames have been sent (I_QUEUE QEMPTY?). If they have, then 5 more are sent.

```
TCLR

1 1 DO_SETID
" THE QUICK BROWN FOX JUMPED OVER THE LAZY DOGS. 1234567890 the quick brown fox
jumped over the lazy dogs. 1234567890 the quick brown fox jumped over the lazy d
ogs. 1234567890  Throughput message "
0 STRING->BUFFER                          ( Create a buffer )

0 SEQ{
      I_QUEUE QEMPTY?                     ( Are all the I-frames sent? )
      IF
          5 0
          DO
              0 SEND_BUFFER               ( Send 5 more )
          LOOP
      ENDIF
  }SEQ

0 STATE_INIT{
      RESET SNRM                          ( Set up the emulation )
      REP_CHAR R=ASCII                    ( Set up the monitor )
  }STATE_INIT                             ( Set initial conditions )

0 STATE{
      OTHER_EVENT
      ACTION{
          0 RUN_SEQ
      }ACTION
  }STATE
```

A
INTRODUCTION TO IBM SNA

This appendix contains a brief introduction to IBM's implementation of SDLC and SNA. For further information, the reader is advised to consult IBM's publication GA27–3136–6, System Network Architecture, Reference Summary; and, publication GA27–3093–2, IBM Synchronous Data Link Control, General Information.

A data link consists of data communication equipment and the communication channel. The combination of DTE's, DCE's, and channel determine what is possible to accomplish with the data link. A number of communication configurations are possible depending on the capabilities of the connected devices.

A.1 SDLC Configurations

There are two types of configurations for SDLC:

- Point–to–point
- Multipoint

These configurations can operate in either two–way alternate (half duplex) or two–way simultaneous mode (full duplex). In two–way alternate mode, stations take turns transmitting. In two–way simultaneous mode, two stations can transmit and receive at the same time.

Point–to–Point

A point-to-point configuration is a data link with two stations. Links can be established on dedicated or dial-up circuits, and can be half duplex or full duplex.

Half duplex channels can be switched (temporary connection) or non-switched (permanent connection).

There are three basic point-to-point data link configurations:

- Half duplex, non-switched
- Full duplex, non-switched
- Half duplex, switched

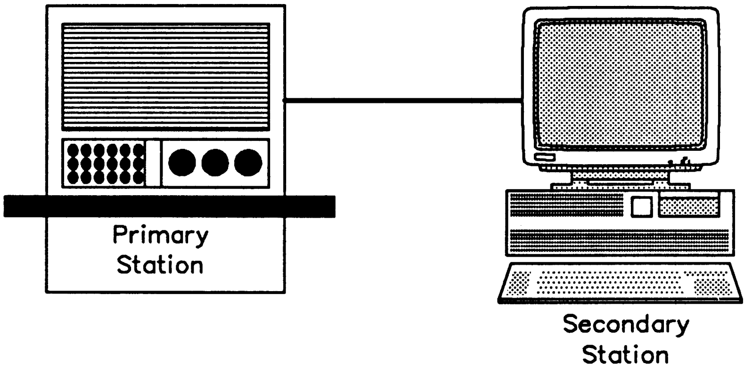


Figure A–1 Point–to–Point Configuration

Multipoint

A multipoint link connects three or more stations.

There are two basic multipoint data link configurations:

- Half duplex, non–switched
- Full duplex, non–switched

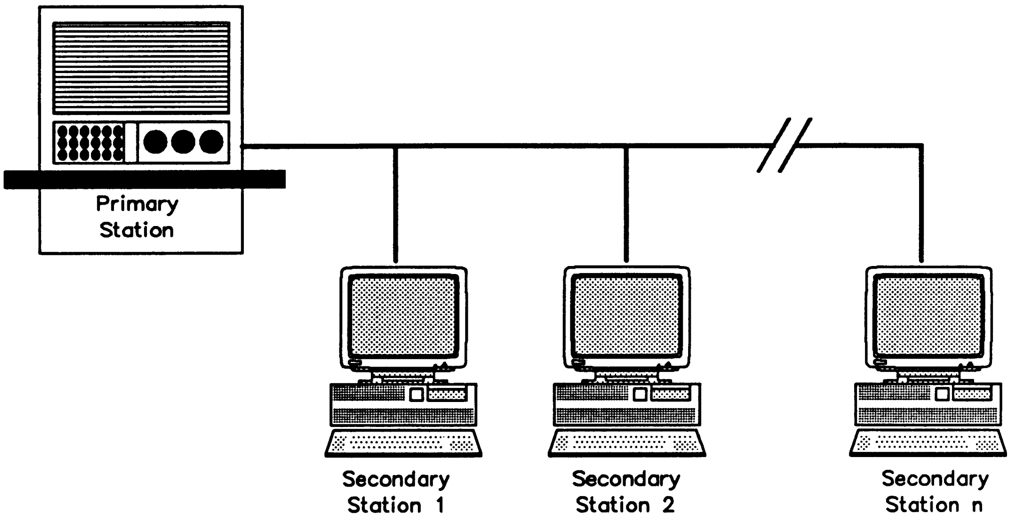


Figure A–2 Multipoint Configuration

A.2 SNA Layers

SNA services are organized into a set of layers. IDACOM's implementation of the SNA Monitor is organized into the following layers:

- physical layer
- data link control layer
- path control layer
- transmission control layer
- data flow control layer
- function management layer
- application layer

SNA Fields

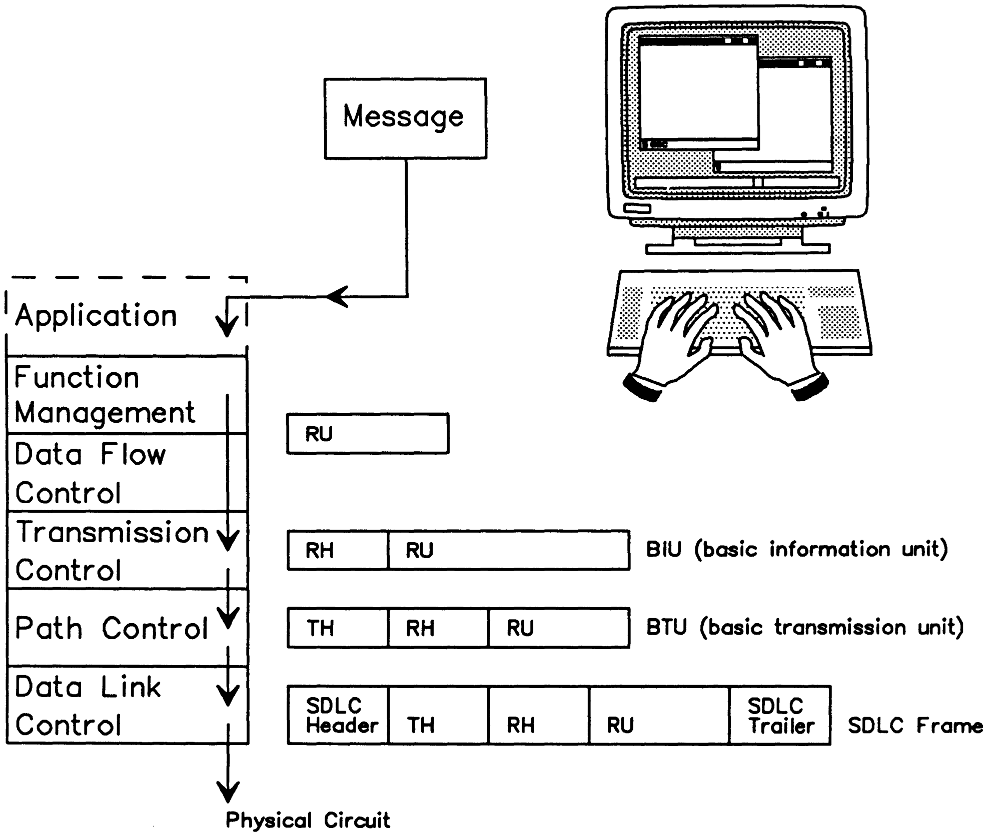


Figure A–3 SNA Layers

IDACOM's SDLC Emulation automatically supports the physical layer and data link control layer. SNA fields can be transmitted but must be user–defined.

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. IDACOM testers support V.28/RS–232C, V.11, V.36/RS–449, and V.35 interfaces.

IDACOM testers also support four methods of clocking: NRZ (standard non–return to zero) external transmit clock, NRZI (non–return to zero inverted), and NRZI with clock. Refer to 'Clocking' in Sections 2 and 9.2 for further information.

A.3 Data Link Control Layer

The data link control layer provides reliable communication between a primary and secondary station. The access procedure used is SDLC (synchronous data link control) and provides mechanisms 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

All SDLC frames contain:

- a start flag;
- an address field; } Link Header
- a control field;
- an information field (optional);
- a frame check sequence (FCS); and } Link Trailer
- a closing flag.

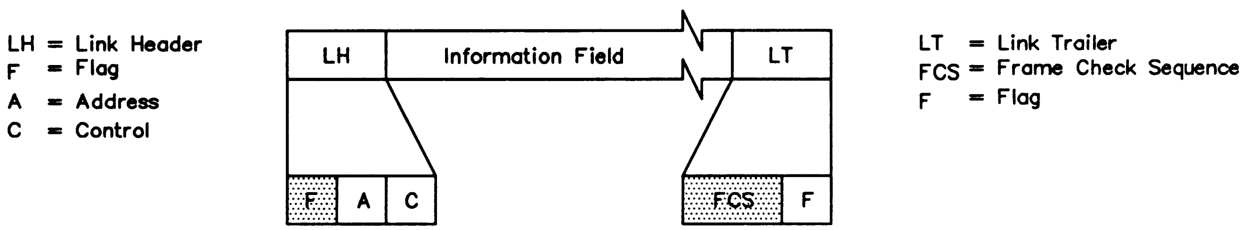


Figure A–4 SDLC Frame Structure

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 SDLC is 0111 1110 (hex 7E).

When receiving, the last zero in the trailing flag can also be the first zero in the next leading flag as illustrated below.

```
       |--leading flag--|
011111101111110
|-trailing flag-|
```

NOTE
Zero bits inserted by the transmission hardware prevent a flag pattern from occurring anywhere else in the frame.

Address

The second byte of the link header is the PU (physical unit) address field. For a primary, this address can be:
- a specific link station address – hex 01 to FE;
- a broadcast address – hex FF; or
- a 'no stations' address – hex 0.

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 be commands or responses.
- control byte contains:
  - an N(S) (send sequence number),
  - an N(R) (receive sequence number) – an acknowledgment of received frames,
  - a P/F (poll/final) bit used to request confirmation, and
- contain additional octets containing SNA 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 (receive 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,
- include:
  - UI (unnumbered information)
  - RIM (request initialization mode)
  - SIM (set initialization mode)
  - DM (disconnect mode)
  - UP (unnumbered poll)
  - RD (request disconnect)
  - DISC (disconnect)
  - UA (unnumbered acknowledgment)
  - SNRM (set normal response mode)
  - FRMR (frame reject)
  - XID (exchange identification)
  - CFGR (configure)
  - TEST (test)
  - BCN (beacon)

Format	Command	Bit 0 (MSB)	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7 (LSB)	Hex Equivalent P/F off	Hex Equivalent P/F on
Information Transfer	Numbered Information Present	R	R	R	P/F	S	S	S	0	X'xx'	X'xx'
Supervisory	RR (receive ready)	R	R	R	P/F	0	0	0	1	X'x1'	X'x1'
	RNR (receive not ready)	R	R	R	P/F	0	1	0	1	X'x5'	X'x5'
	REJ (reject)	R	R	R	P/F	1	0	0	1	X'x9'	X'x9'
Unnumbered	UI (unnumbered information)	0	0	0	P/F	0	0	1	1	X'03'	X'13'
	RIM (request initialization mode)	0	0	0	F	0	1	1	1	X'07'	X'17'
	SIM (set initialization mode)	0	0	0	P	0	1	1	1	X'07	X'17'
	DM (disconnect mode)	0	0	0	F	1	1	1	1	X'0F'	X'1F'
	UP (unnumbered poll)	0	0	1	P	0	0	1	1	X'23'	X'33'
	RD (request disconnect)	0	1	0	F	0	0	1	1	X'43'	X'53'
	DISC (disconnect)	0	1	0	P	0	0	1	1	X'43'	X'53'
	UA (unnumbered acknowledgement)	0	1	1	F	0	0	1	1	X'63'	X'73'
	SNRM (set normal response mode)	1	0	0	P	0	0	1	1	X'83'	X'93'
	FRMR (frame reject)	1	0	0	F	0	1	1	1	X'87'	X'97'
	XID (exchange identification)	1	0	1	P/F	1	1	1	1	X'AF'	X'BF'
	CFGR (configure)	1	1	0	P/F	0	1	1	1	X'C7'	X'D7'
	TEST (test)	1	1	1	P/F	0	0	1	1	X'E3'	X'F3'
	BCN (beacon)	1	1	1	F	1	1	1	1	X'EF'	X'FF'

NOTES: P = Poll bit
F = Final bit
RRR = Nr (receive count)
SSS = Ns (send count)

Table A–1 SDLC Commands and Responses – Modulo 8

Format	Command	Binary Configuration																Hex Equivalent
	Bits	MSB 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	LSB 15	
Information Transfer	Numbered Information Present	S	S	S	S	S	S	S	0	R	R	R	R	R	R	R	P/F	X'xxxx'
Supervisory	RR (receive ready)	0	0	0	0	0	0	0	1	R	R	R	R	R	R	R	P/F	X'01xx'
	RNR (receive not ready)	0	0	0	0	0	1	0	1	R	R	R	R	R	R	R	P/F	X'05xx'
	REJ (reject)	0	0	0	0	1	0	0	1	R	R	R	R	R	R	R	P/F	X'09xx'

NOTES: P = Poll bit
F = Final bit
RRR = Nr (receive count)
SSS = Ns (send count)

Table A–2 SDLC Commands and Responses – Modulo 128

In Tables A–1 and A–2, frames containing a P bit are commands and are issued by a primary. Frames containing an F bit are responses and are issued by a secondary. Frames containing P/F can be issued by either a primary or a secondary.

A.4 Path Control

Path control adds a TH (transmission header) to the BIU, creating a BTU (basic transmission unit).

SNA path control provides the following major services:

- routing data from node to node;
- segmenting and blocking messages;
- sequencing messages sent over multiple transmission links;
- controlling transmission priorities; and
- handling virtual route pacing.

SNA defines six types of TH (transmission header). These are FID0, FID1, FID2, FID3, FID4, and FIDF. The format and length of each type is determined by the FID (format identifier) which is located in byte 0 of the transmission header.

The FID type depends on the type(s) of nodes involved in the transmission:

FID type 0 (length equals 10 bytes) is used for traffic involving non–SNA devices between adjacent subarea nodes when either or both nodes do not support explicit route and virtual route protocols.

FID type 1 (length equals 10 bytes) is used for traffic between adjacent subarea nodes when either or both nodes do not support explicit route and virtual route protocols.

FID type 2 (length equals 6 bytes) is used for traffic between a subarea node and an adjacent type 2.0 or 2.1 peripheral node.

FID type 3 (length equals 2 bytes) is used for traffic between a subarea node and an adjacent type 1 peripheral node.

FID type 4 (length equals 26 bytes) is used for traffic between adjacent subarea nodes when both nodes support explicit route and virtual route protocols.

FID type F (length equals 26 bytes) is used for certain commands (for example, for transmission group control) sent between adjacent subarea nodes when both nodes support explicit route and virtual route protocols.

Figures A-5 to A-12 show the formats and descriptions of the bytes in each FID type.

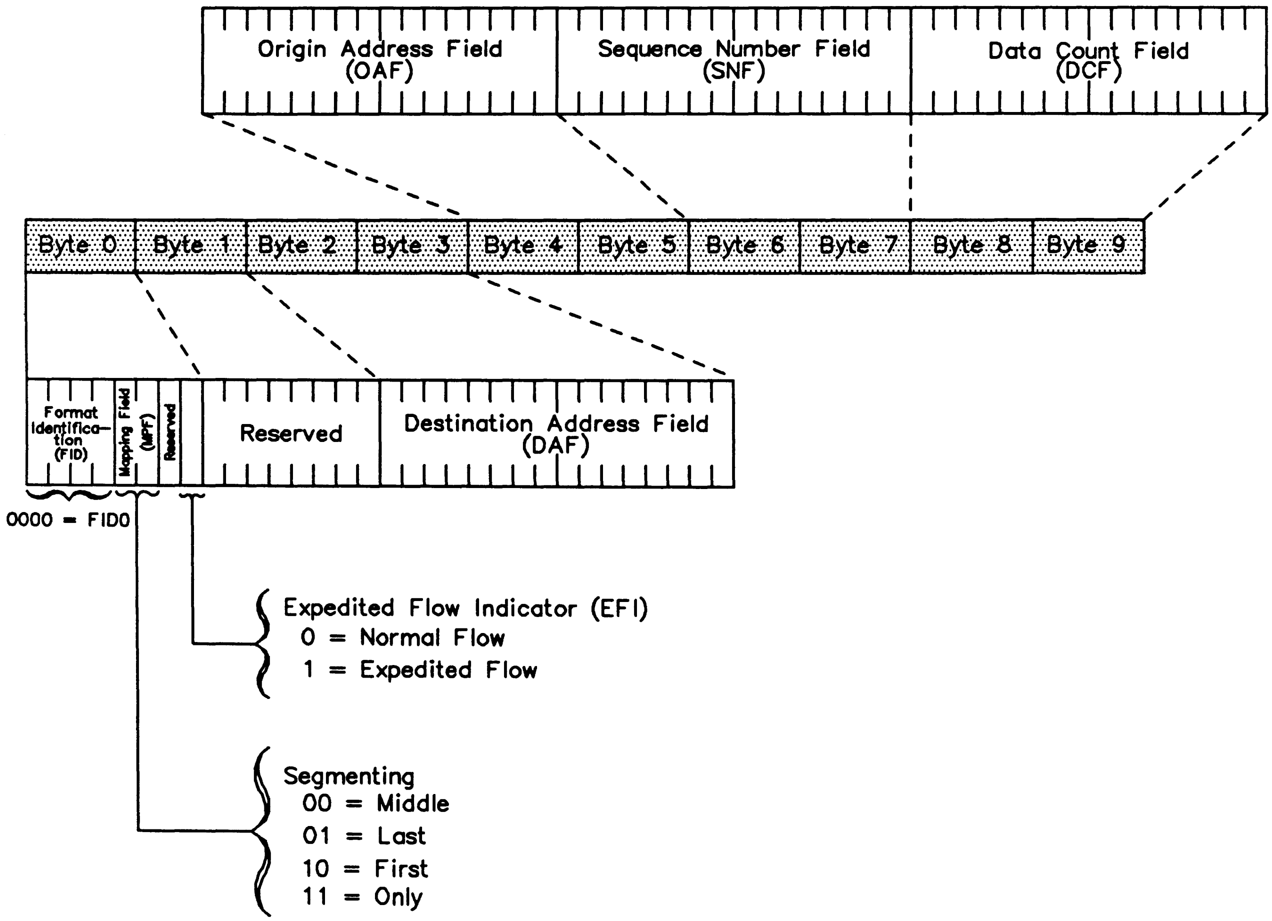


Figure A-5 Transmission Header for FID Type 0

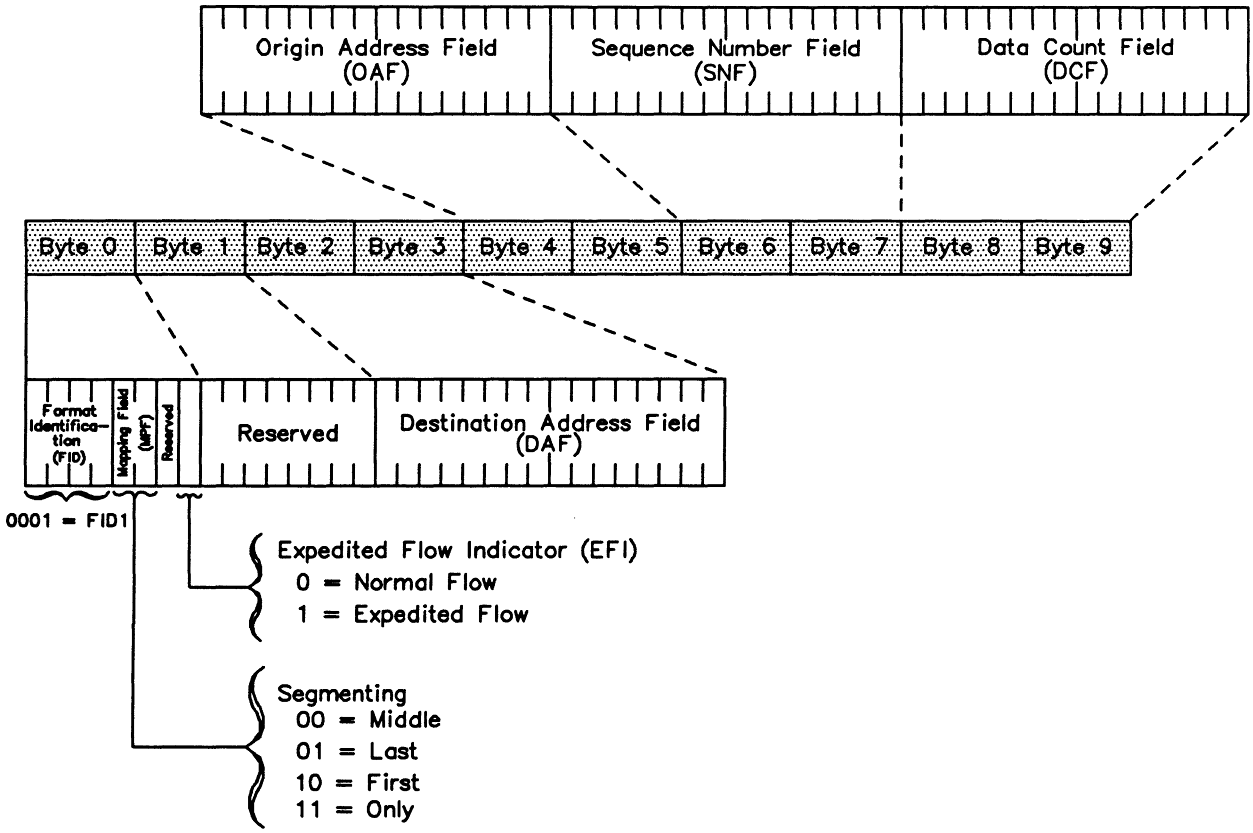


Figure A–6 Transmission Header for FID Type 1

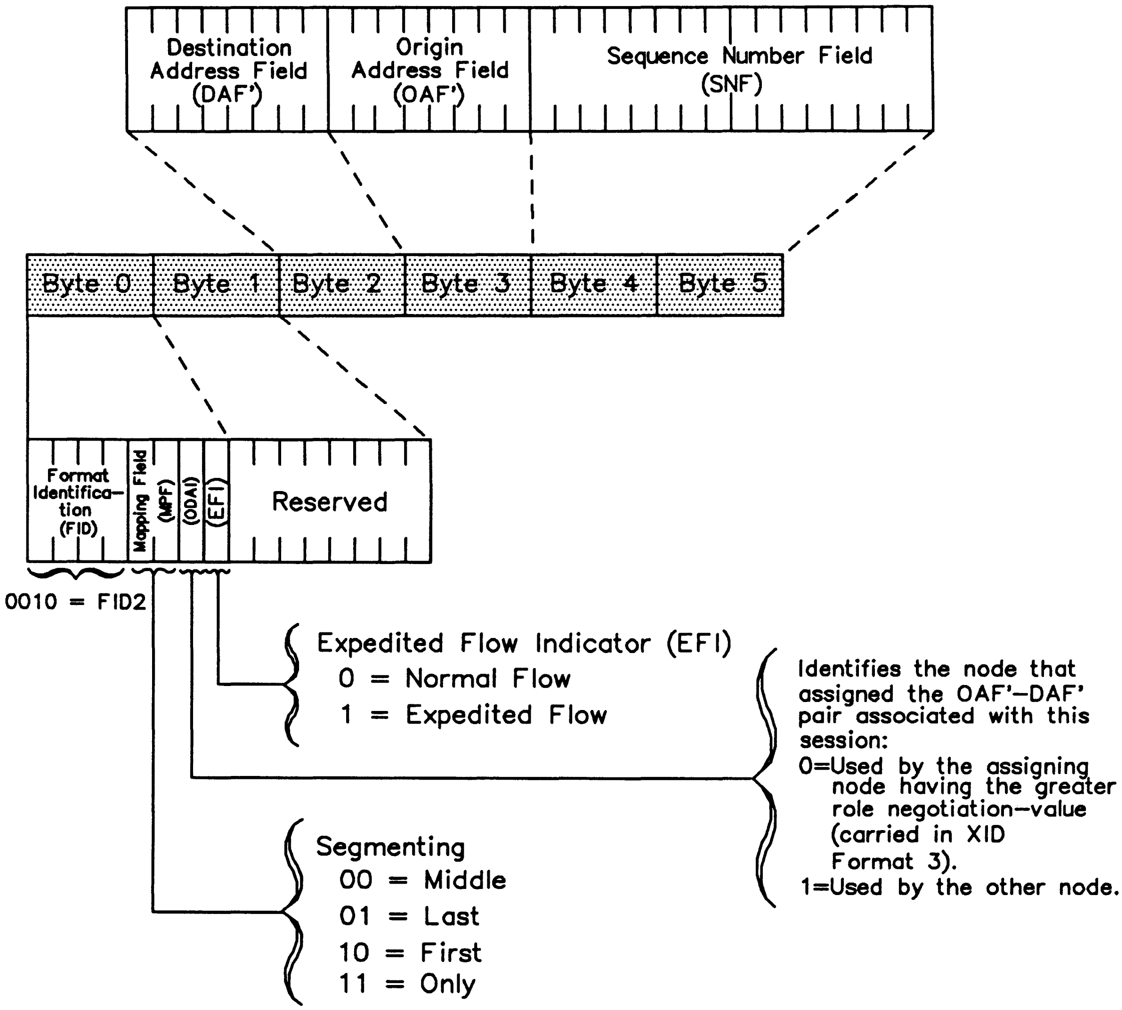


Figure A–7 Transmission Header for FID Type 2

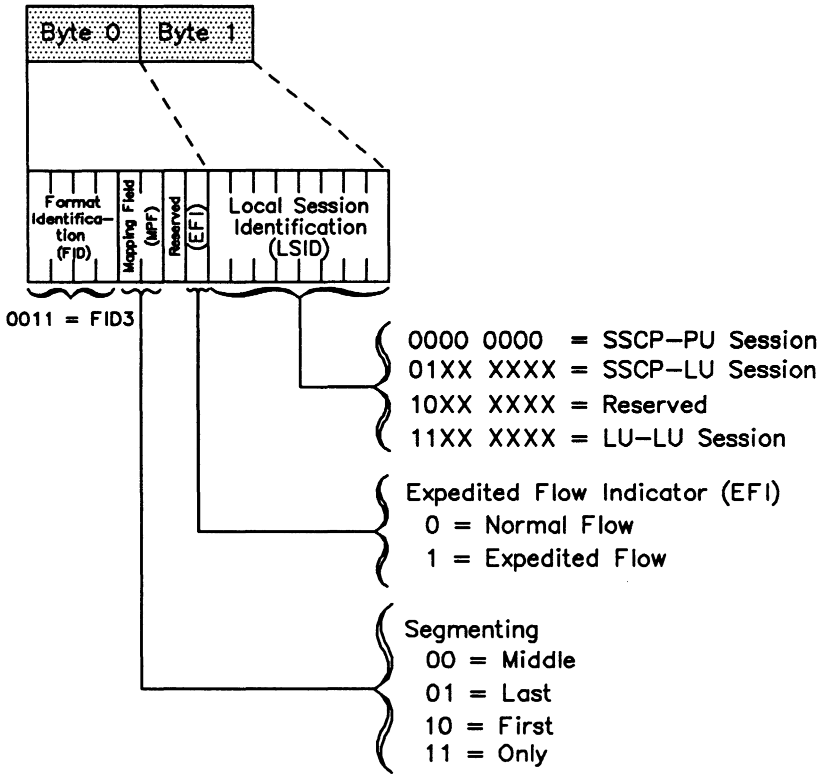


Figure A-8 Transmission Header for FID Type 3

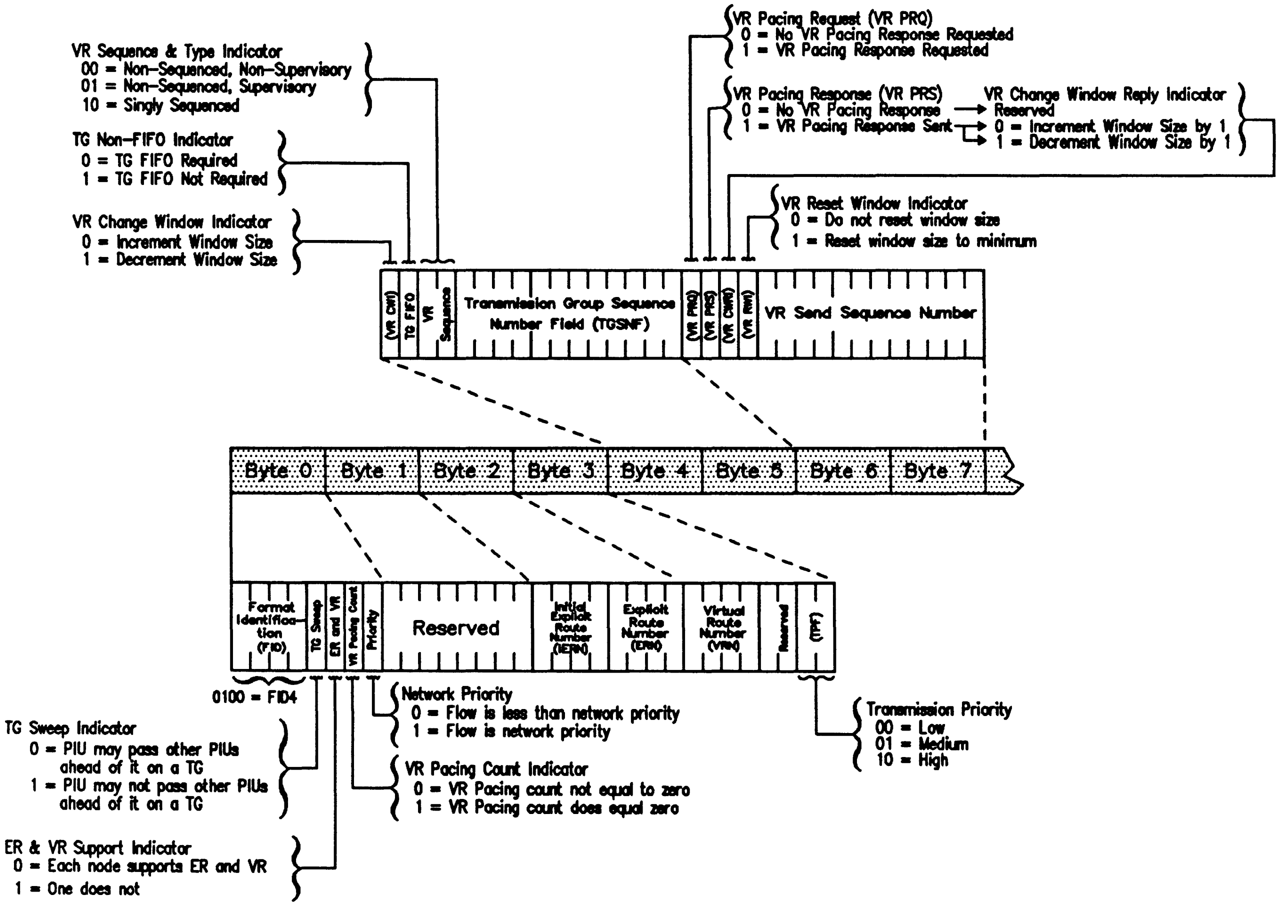


Figure A-9 Transmission Header for FID Type 4 (Bytes 0 to 7)

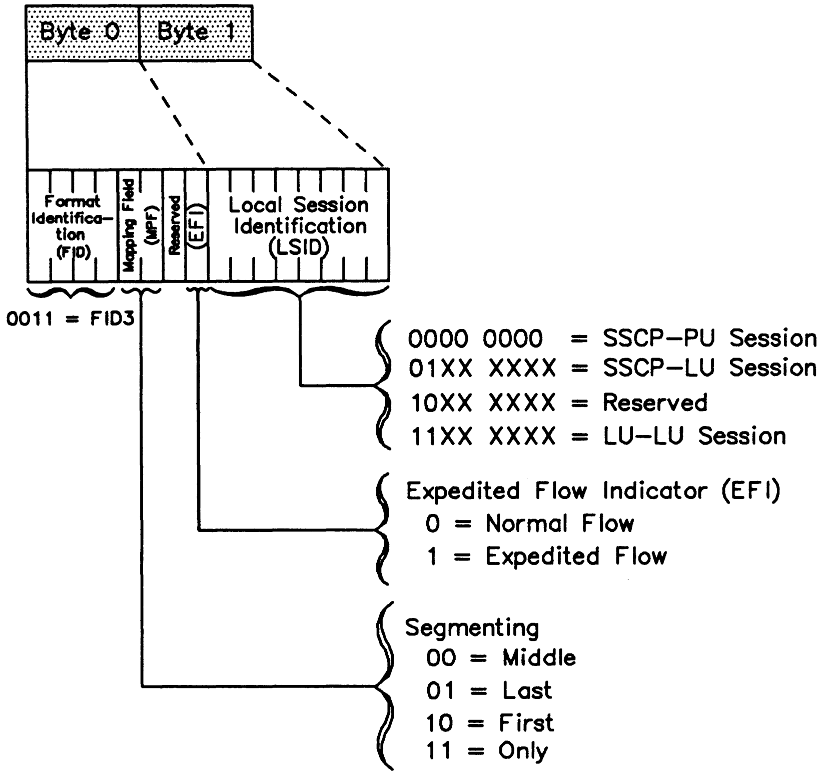


Figure A-8 Transmission Header for FID Type 3

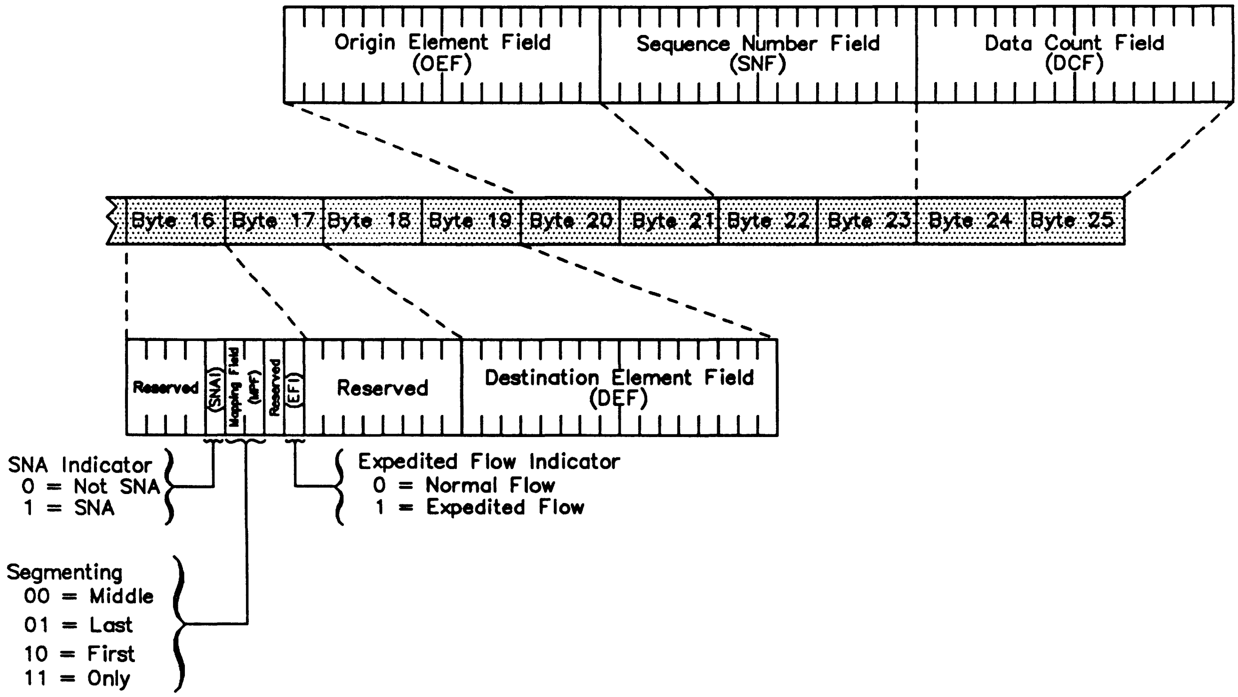


Figure A-11 Transmission Header for FID Type 4 (Bytes 16 – 25)

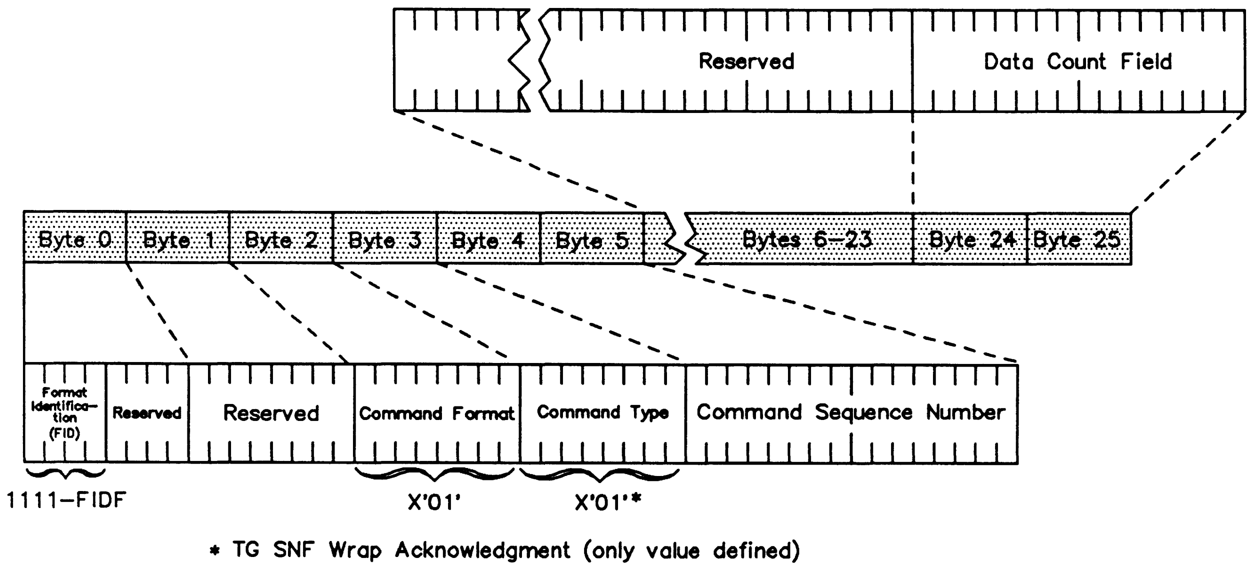


Figure A-12 Transmission Header for FID Type F

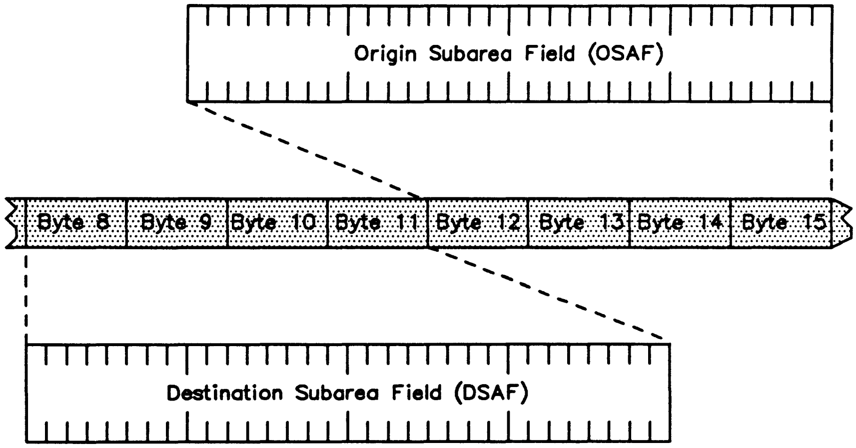


Figure A-10 Transmission Header for FID Type 4 (Bytes 8-15)

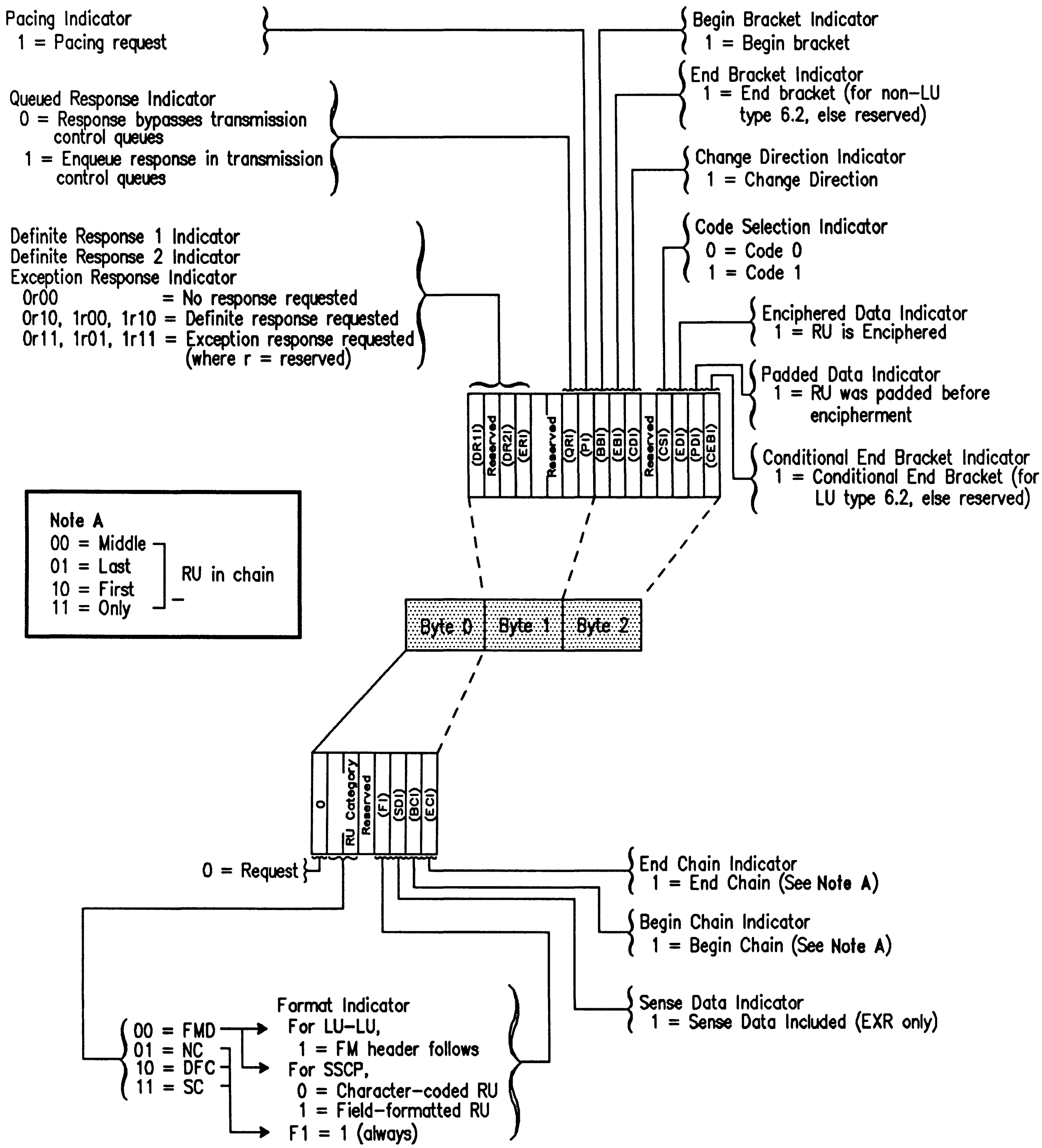


Figure A–14 Request Header

A.5 Transmission Control

When a message and its associated parameters are sent down from data flow control, transmission control uses the information supplied in the parameters to construct an RH (request/response header).

The following major functions are performed by the transmission control:

- Sequencing
- Cryptography
- Session – level pacing

The RH (request/response header) is a 3 byte field immediately following the transmission header. In some cases, the transmission control layer creates a BIU (basic information unit) that contains only an RH and has no associated RU. These BIU's are used to perform control functions.

Path control can segment the BIU to form several PIU's (path information units) as shown in Figure A–13. When the BIU is segmented, the first segment contains the request/response header.

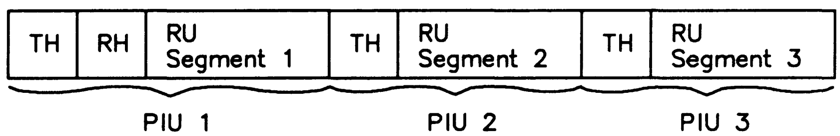


Figure A–13 BIU Format – Multiple PIU's

If bit 0 in byte 0 of the RH equals zero, the RH is a request header and the associated RU is a request unit. If bit 0 of byte 0 of the RH equals 1, the RH is a response header and the associated RU is a response unit. Figure A–14 shows the formats and descriptions of the bytes in a request header. Figure A–15 shows the formats and descriptions of the bytes in a response header.

A.6 Data Flow Control

Data flow control communicates with both function management and transmission control by passing RU's (request/response units) and parameters. Data flow control functions include:

- processing a series of messages as a chain;
- processing a series of messages and responses as a bracket;
- determining and controlling the send/receive mode;
- assigning sequence numbers to RU's;
- interrupting data flow on request; and
- processing session status and error recovery.

A.7 Function Management

Function management constructs an RU (request/response unit) from the message created at the application layer. The RU is passed down through the data flow layer to transmission control.

SD (Sense Data)

If the sense data indicator in the RH (request/response header) is set to 1, 4 bytes of sense data are inserted after the RH and prior to the RU (request/response unit). The format of this field is shown in Figure A-16.

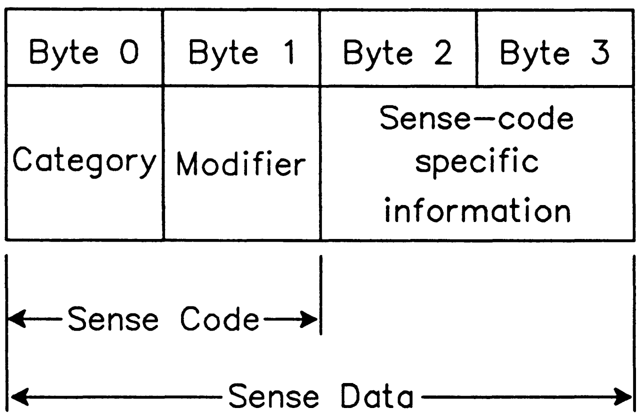


Figure A-16 Sense Data Format

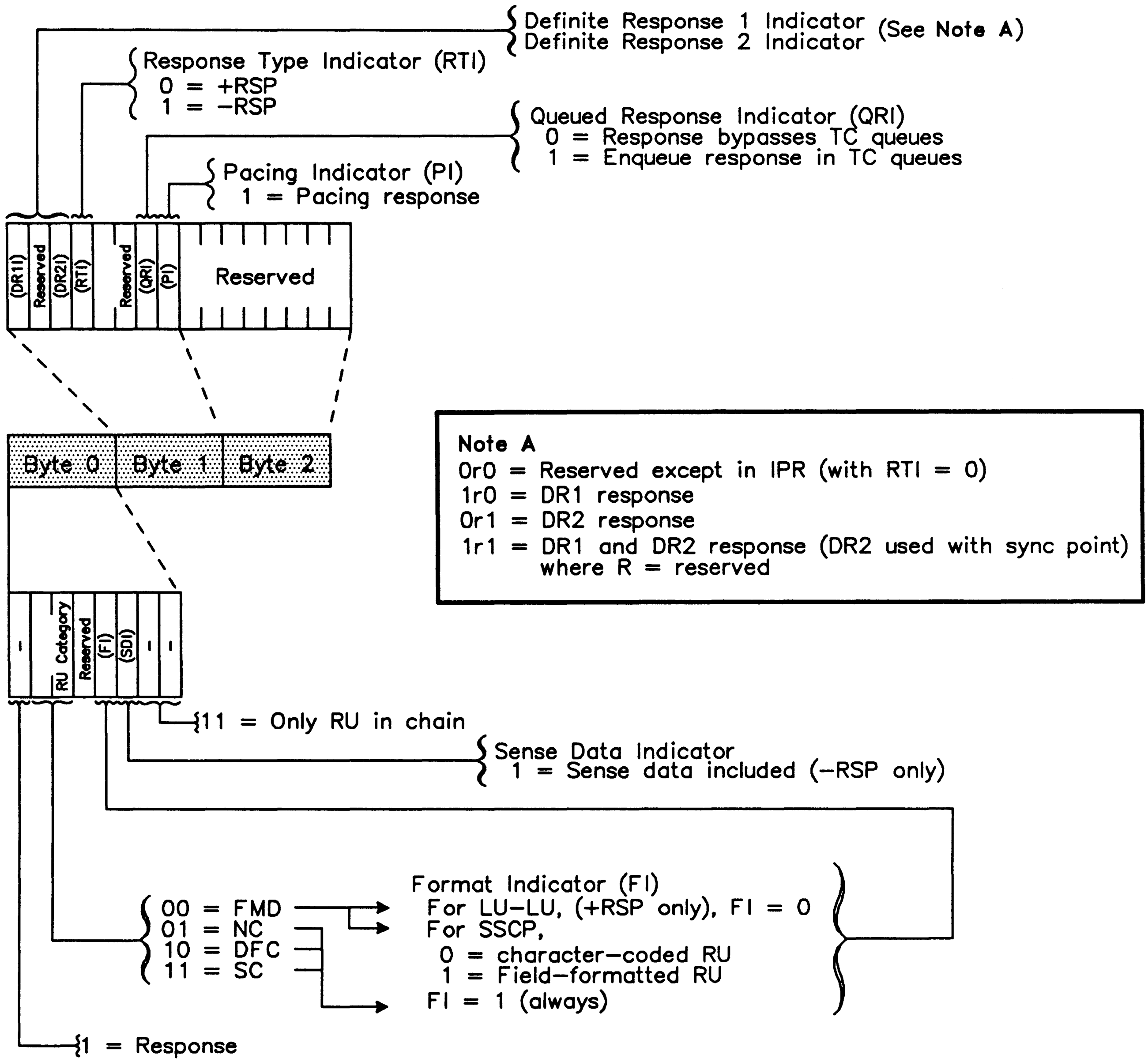


Figure A–15 Response Header

Together, the category byte 0, the modifier byte 1, and the sense code specific bytes 2 and 3 hold the sense data defined for the exception condition that has occurred.

Refer to IBM's publication GA27–3136–B, System Network Architecture, Reference Summary, Chapter 9, Sense Data for more information on sense–code specific information.

RU (Request/Response Unit)

Request/response units follow the RH if sense data is not present. If sense data is present, the RU follows the sense data. For further information, refer to IBM's publication GA27–3136–B, System Network Architecture, Reference Summary, Chapter 4 – Requests and Chapter 5 – Responses.

A.8 Application Layer

A user operating at the application layer formulates a message and passes it to the function management layer of SNA.

B
DATA FORMATS

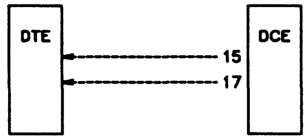
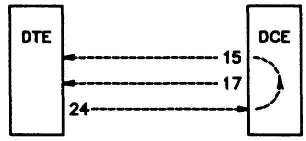
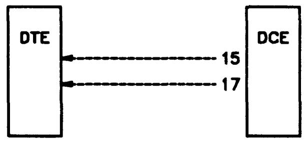
Clocking Mode	Encoding Scheme	Clocking Source
NRZ with Clock	NRZ	
External Tx Clock	NRZ	
NRZI With Clock	NRZI	
NRZI	NRZI	Clock speed is extracted from the data signal.
NRZ (Non-Return to Zero)	A 1-bit maps to a mark signal. A 0-bit maps to a space signal.	
NRZI (Non-Return to Zero Inverted)	A 1-bit maps to no transition. A 0-bit maps to a transition.	
15 – Transmit clock from DCE (DCE provided) CCITT circuit 114 17 – Receive clock from DCE (DCE provided) CCITT circuit 115 24 – Transmit clock to DCE (DTE provided) CCITT circuit 11 The pin numbers shown are for the RS-232C interface.		

Table B-1 Clocking Modes

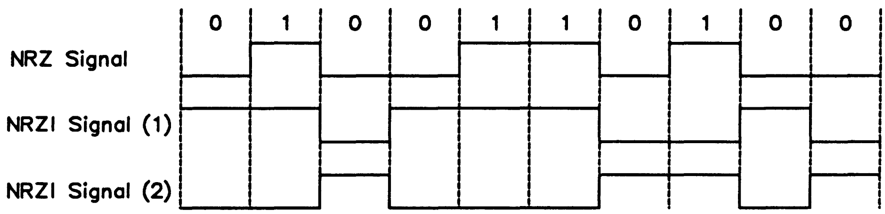


Figure B-1 NRZ and NRZI Data Encoding

BIU Segment
The portion of a basic information unit (BIU) that is contained within a path information unit (PIU). It consists of either a request/response header (RH) followed by either all or part of a request/response unit (RU), or only part of an RU.

Bracket
A series of related normal–flow chains and their associated responses can be grouped together into a larger unit called a bracket. A bracket must be completed before another bracket can be started. Examples of brackets are data base inquiries/replies, update transactions, and remote job entry output sequences to work stations. See also begin bracket, end bracket, RU chain.

Bracket Protocol
A data flow control protocol in which exchanges between the two LU–LU half sessions are achieved through the use of brackets, with one LU designated at session activation as the first speaker and the other LU as the bidder. The bracket protocol involves bracket initiation and termination rules.

BTU
See basic transmission unit.

CANCEL
Cancel.

CDCINIT
Cross–domain control initiate.

CDI
Change direction indicator. See also change direction protocol.

CDINIT
Cross–domain initiate.

CDRM
Cross–domain resource manager.

CDSESSEND
Cross–domain session end.

CDSESSSF
Cross–domain session setup failure.

CDSESSST
Cross–domain session started.

CDSESSTF
Cross–domain session takedown failure.

CDTAKED
Cross–domain takedown.

CDTAKEDC
Cross–domain takedown complete.

CDTERM
Cross–domain terminate.

CEBI
Conditional end bracket indicator.

Chain
See RU chain.

Channel
See data channel.

Change Direction Protocol
A data flow control protocol in which the sending logical unit (LU) stops sending normal flow requests, signals this fact to the receiving LU using the change direction indicator (in the request header of the last request of the last chain), and prepares to receive requests.

CHASE
Chase.

CINIT
Control initiate.

CLEANUP
Clean up session.

CLEAR
Clear.

Cluster Controller Node
A peripheral node that can control a variety of devices.

Command
1. Any field set in the transmission header (TH), request header (RH), and sometimes portions of a request unit, that initiates an action or that begins a protocol; for example:
   a. bind session (session control request unit), a command that activates an LU–LU session,
   b. the change direction indicator in the RH of the last RU of a chain,
   c. the virtual route reset window indicator in an FID4 transmission header.
2. Loosely, a request unit.
3. In SDLC, the control information (in the C–field of the link header) sent from the primary station to the secondary station.

C
SDLC/SNA TERMINOLOGY

A
Address byte of the SDLC link header.

ABCONN
Abandon connection.

ABCONNOUT
Abandon connect out.

ACTCDRM
Activate cross–domain resource manager.

ACTCONNIN
Activate connection in.

ACTLINK
Activate link.

ACTLU
Activate logical unit.

ACTPU
Activate physical unit.

ACTTRACE
Activate trace.

ADDLINK
Add link.

ADDLINK STA
Add link station.

ANA
Assign network addresses.

Basic Information Unit (BIU)
The unit of data and control information that is passed between half–sessions. It consists of a request/response header (RH) followed by a request/response unit (RU).

Basic Link Unit (BLU)
The unit of data and control information transmitted over a link by data link control.

Basic Transmission Unit (BTU)
The unit of data and control information passed between path control components. A BTU can consist of one or more path information units (PIU's). See also path information unit.

BB
See begin bracket.

BBI
Begin bracket indicator.

BC
See begin chain.

BCI
Begin chain indicator.

Begin Bracket (BB)
The value (binary 1) of the begin bracket indicator in the request header (RH) of the first request in the first chain of a bracket; the value denotes the start of a bracket. Contrast with end bracket. See also bracket.

Begin Chain (BC)
The value (binary 1) of the begin chain indicator in the request header (RH) of the first of the RU chain; the value denotes the start of the chain. Contrast with end chain. See also RU chain.

BID
Bid

BIND
Bind session.

BINDF
Bind failure.

BIS
Bracket initiation stopped.

BIU
See basic information unit.

Data Flow Control (DFC)
A request/response unit (RU) category used for requests and responses exchanged between the data flow control layer in one half session and the data flow control layer in the session partner.

Data Flow Control (DFC) Layer
The layer within a half session that:

1. Controls whether the half session can send, receive, or concurrently send and receive request units (RUs).
2. Groups related RUs into RU chains.
3. Delimits transactions via the bracket protocol.
4. Controls the interlocking of requests and responses in accordance with control modes specified at session activation.
5. Generates sequence numbers.
6. Correlates requests and responses.

Data Link
Synonym for link.

Data Link Control (DLC) Layer
The layer that consists of the link stations that schedule data transfer over a link between two nodes and perform error control for the link. Examples of data link control are SDLC for serial–by–bit link connection and data link control for the System/370 channel.

Data Stream
A continuous stream of data elements being transmitted, or intended for transmission, in character or binary–digit form, using a defined format.

DCE
Data circuit terminating equipment.

DCF
See data count field.

DEF
See destination element field.

Definite Response
A value in the form–of–response–requested field of the request header. The value directs the receiver of the request to return a response unconditionally, whether positive or negative, to that request. Contrast with exception response, no response.

DELETENR
Delete network resource.

Destination Address Field (DAF)
A field in an FID0 or FID1 transmission header that contains the network address of the destination. See also destination address field prime (DAF'), destination element field (DEF), destination subarea field (DSAF), format identification (FID) field, local session identification (LSID). Contrast with origin address field (OAF).

Destination Address Field Prime (DAF')
A field in an FID2 transmission header that contains the local address of the destination network addressable unit (NAU). See also destination address field (DAF), format identification (FID) field. Contrast with origin address field prime (OAF').

Destination Element Field (DEF)
A field in an FID4 transmission that contains an element address which, combined with the subarea address in the destination subarea field (DSAF), gives the complete network address of the destination network addressable unit (NAU). See also format identification (FID) field. Contrast with origin element field (OEF).

Destination Subarea Field (DSAF)
A field in an FID4 transmission header that contains a subarea address which, combined with the element address in the destination element field (DEF), gives the complete network address of the destination network addressable unit (NAU). See also format identification (FID field. Contrast with origin subarea field (OSAF).

DFC
See data flow control.

DISC
SDLC disconnect frame.

DISCONTACT
Discontact.

DISPSTOR
Display storage.

DLC
See data link control.

DM
SDLC disconnected mode frame.

DR1I
Definite response 1 indicator.

DR2I
Definite response 2 indicator.

Communication Adaptor
An optional hardware feature, available on certain processors, that permits communication lines to be attached to processors.

Communication Controller
A type of communication control unit whose operations are controlled by one or more programs stored and executed in the unit; for example, the IBM 3705 Communications Controller.

Configuration Services
One of the types of network services in the system services control point (SSCP) and in the physical unit (PU); configuration services activate, deactivate, and maintain the status of physical units, links, and link stations. Configuration services also shut down and restart network elements and modify path control routing tables and address-transformation tables. See also maintenance services, management services, network services, physical unit control point, session services, SSCP.

Communication Controller Node
A term used to refer to a subarea node containing no system services control point (SSCP).

CONNOUT
Connect out.

CONTACT
Contact.

CONTACTED
Contacted.

Control Point
A physical unit control point (PUCP) or a system services control point (SSCP).

CRC
Cyclic redundancy checking.

Cross-Domain
Pertaining to control or resources involving more than one domain.

CRV
Cryptography verification.

Cryptographic Key
In systems using the data encryption standard (DES) algorithm, a 64 bit value (containing 56 independent bits and 8 parity bits) provided as input to the algorithm in determining the output of the algorithm.

Cryptographic Session
An LU-LU session in which a function management data (FMD) request can be enciphered before it is transmitted and deciphered after it is received.

Cryptography
The transformation of data to conceal its meaning.

CSI
Code selection indicator.

CTERM
Control terminate.

DACTCDRM
Deactivate cross-domain resource manager.

DACTCONNIN
Deactivate connect in.

DACTLINK
Deactivate link.

DACTLU
Deactivate logical unit.

DACTPU
Deactivate physical unit.

DACTTRACE
Deactivate trace.

DAF
See destination address field.

DAF'
See destination address field prime.

Data Channel
A device that connects a processor and main storage with I/O control units.

Data Circuit
Synonym for link connection.

Data Count Field (DCF)
A binary count of the number of bytes in the basic information unit (BIU) or BIU segment associated with the transmission header (TH).

Data Encrypting Key
A key used to encipher and decipher data transmitted in a session that uses cryptography.

Expedited Flow
A data flow designated in the transmission header (TH) that is used to carry network control, session control, and various data flow control request/response units (RUs); the expedited flow is separate from the normal flow (which carries primarily end–user data) and can be used for commands that affect the normal flow. Contrast with normal flow.

NOTE
The normal and expedited flows move in both the primary to secondary and secondary to primary directions. Request and responses on a given flow (normal or expedited) usually are processed sequentially within the path, but the expedited flow traffic can be moved ahead of the normal flow traffic within the path at queuing points in the half sessions and for half session support in boundary functions.

Explicit Route (ER)
The path control network components, including a specific set of one or more transmission groups, that connect two subarea nodes. An explicit route is identified by an origin subarea address, a destination subarea address, an explicit route number, and a reverse explicit route number. See also path, virtual route.

Explicit Route Length
The number of transmission groups in an explicit route.

EXR
See exception request.

EXSLOW
Exiting slowdown.

F
See flag.

FCS
SDLC frame check sequence.

FID
Format identification. See format identification field.

FIFO
First in, first out.

Flag
Sync character of hexadecimal 7E.

Flow Control
The process of managing the rate at which data traffic passes between components of the network. Flow control optimizes the rate of flow of message units with minimum congestion in the network; that is, to neither overflow the buffers at the receiver or at intermediate routing nodes, nor leave the receiver waiting for more message units. See also pacing, session level pacing, virtual route (VR) pacing.

FM
Function management.

FMD
See function management data.

FMD Services Layer
The layer within a half session that routes FMD requests and responses to particular NAU services manager components and that provides session network services or session presentation services, depending on the type of session.

FMH
See function management header.

FNA
Free network address.

Format Identification (FID) Field
A field in each transmission header (TH) that indicates the format of the TH; that is, the presence or absence of certain fields. Transmission header formats differ in accordance with the types of nodes between which they pass.

Frame
See basic link unit.

FRMR
SDLC frame reject.

Function Management Data (FMD)
An RU category used for end user data exchanged between logical units (LUs) and for requests and responses exchanged between network services components of LUs, PUs, and SSCPs.

Function Management Data Services
A generic term for session network services and session presentation services, both of which process FMD requests and responses.

DSAF
See destination subarea field.

DSRLST
Direct search list.

DTE
Data terminal equipment.

DUMPFINAL
Dump final.

DUMPINIT
Dump initial.

DUMPTEXT
Dump text.

EB
See end bracket.

EBCDIC
Extended binary coded decimal interchange code.

EBI
End bracket indicator. See also end bracket.

EC
See end chain.

ECHOTEST
Echo test.

ECI
End chain indicator. See also end chain.

ED
The value (binary 1) of the enciphered data indicator in the request header (RH) denotes that the request unit (RU) is enciphered. See also cryptography.

EDI
Enciphered data indicator.

EFI
Expedited flow indicator. See also expedited flow.

End Bracket
The value (binary 1) of the end bracket indicator in the request header (RH) of the first request of the last chain of a bracket; the value denotes the end of the bracket. Contrast with begin bracket. See also bracket.

End Chain
The value (binary 1) of the end chain indicator in the request header (RH) of the last of the RU chain; the value indicates the end of the chain. Contrast with begin chain. See also RU chain.

ER
See explicit route.

ER–INOP
Explicit route inoperative.

ER_TESTED
Explicit route tested.

ERI
Exception response indicator. See also exception response.

ERN
Explicit route number.

ESLOW
Entering slowdown.

Exception Request (EXR)
A request that replaces another message unit in which an error has been detected.

NOTE
The exception request contains a 4 byte sense field that identifies the error in the original message unit and, except for some path errors, is sent to the destination of the original message unit; if possible, the sense data is returned in a negative response to the originator of the replaced message unit.

Exception Response
A value in the form of response requested field of a request header: the receiver is requested to return a response only if the request is unacceptable as received or cannot be processed; that is, a negative response, but not a positive response, could be returned.

EXECTEST
Execute test.

Local Address
An address used in a peripheral node in place of a network address and transformed to or from a network address by the boundary function in a subarea node. See also network address.

Local Session Identification
A field in an FID3 transmission header that contains an indication of the type of session (SSCP-PU, SSCP-LU, or LU-LU) and the local address of the peripheral logical unit (LU) or physical unit (PU).

Logical Unit (LU)
A port through which an end user accesses the SNA network in order to communicate with another end user and through which the end user accesses the functions provided by system services control points (SSCPs). An LU can support at least two sessions – one with an SSCP, and one with another logical unit and can be capable of supporting many sessions with other logical units.

LSA
Lost subarea.

LSID
See local session identification.

LT
See link trailer.

LU
See logical unit.

LUSTAT
Logical unit status.

Maintenance Services
One of the types of network services in system services control points (SSCP's) and physical unit (PU's). Maintenance services provide facilities for testing links and nodes and for collecting and recording error information. See also configuration services, management services, network services, session services.

Management Services
One of the types of network services in system services control points (SSCPs) and logical units (LUs). Management services forward requests for network data, such as error statistics, and deliver the data in reply. See also configuration services, maintenance services, network services, and session services.

Message Segment
That portion of a message that is contained within a single request unit.

Message Unit
A generic term for the unit of data processed by any layer; for example, a basic information unit (BIU), a path information unit (PIU), a request/response unit (RU).

NA
See network address.

NAU
See network addressable unit.

NAU Services
The functions provided by the NAU services manager layer and the FMD services layer.

NAU Services Manager Layer
The layer that:

1. Controls network operations via LU-LU, SSCP-LU, SSCP-PU, and SSCP-SSCP sessions.
2. Coordinates end user interactions on LU-LU sessions. See also configuration services, session services, maintenance services, management services.

NC
See network control.

NC-ACTVR
NC activate virtual route.

NC-DACTVR
NC deactivate virtual route.

NC-ER-ACT
NC explicit route activate.

NC-ER-ACT-REPLY
NC explicit route activate reply.

NC-ER-INOP
NC explicit route inoperative

NC-ER-OP
NC explicit route operative

NC-ER-TEST
NC explicit route test.

NC-ER-TEST-REPLY
NC explicit route test reply.

NC-IPL-ABORT
NC IPL abort.

NC-IPL-FINAL
NC IPL final.

Function Management Header
One or more headers, optionally present in the leading request units (RUs) of an RU chain, that allow one half session in an LU-LU session to:

1. Select a destination as the session partner and control the way the end user data it sends is handled at the destination.
2. Change the destination or the characteristics of the data during the session.
3. Transmit between session partners status or user information about the destination (for example, whether it is a program or a device).

Half Session
A component that provides FMD services, data flow control, and transmission control for one of the sessions of a network addressable unit (NAU).

HDX
Half duplex data flow.

I
SDLC information frame.

ID
Identification

IERN
Initial explicit route number.

INIT-OTHER
Initiate-other.

INIT-OTHER-CD
Initiate-other cross-domain.

INIT-SELF
Initiate-self.

Initiation
See LU-LU session initiation. See also session initiation request.

INITPROC
Initiate procedure.

INOP
Inoperative.

IPLFINAL
IPL final.

IPLINIT
IPL initial.

IPLTEXT
IPL text.

Layer
A grouping of related functions that are logically separate from the functions in other layers; the implementation of the functions in one layer can be changed without affecting functions in other layers. See also NAU services manager layer, FMD services layer, data flow control layer, transmission control layer, path control layer, data link control layer.

LCP
Lost control point.

LDREQD
Load required.

LH
See link header.

LIFO
Last in, first out.

Link
The combination of the link connection and the link stations joining network nodes; for example:

1. A System/370 channel and its associated protocols.
2. A serial-by-bit connection under the control of synchronous data link control (SDLC).

Synonymous with data link.

NOTE
A link connection is the physical medium of transmission; for example, a telephone wire or a microwave beam. A link includes the physical medium of transmission, the protocol, and associated communication devices and programming; it is both logical and physical.

Link Connection
The physical equipment providing two-way communication between one link station and one or more other link stations; for example, a communication line and data circuit terminated equipment (DCE). Synonymous with data circuit.

Link Header
Control information for data link control at the beginning of a basic link unit (BLU).

Link Station
The combination of hardware and software that allows a node to attach to and provide control for a link.

Link Trailer
Control information for data link control at the end of a basic link unit (BLU).

NS-IPL-INIT
NS IPL initial.

NS-IPL-TEXT
NS IPL text.

NS-LSA
NS lost subarea.

NSPE
NS procedure error.

OAF
See origin address field.

OAF'
See origin address field prime.

OEF
See origin element field.

Origin Address Field (OAF)
A field in an FID0 or FID1 transmission header that contains the address of the originating network addressable unit. Contrast with destination address field. See also format identification (FID) field, local session identification (LSID), origin address field prime (OAF'), origin element field (OEF), origin subarea field (OSAF).

Origin Element Field (OEF)
A field in an FID4 transmission header that contains an element address, which combined with the subarea address in the origin subarea field (OSAF), gives the complete network address of the originating network addressable unit (NAU). Contrast with destination element field (DEF). See also format identification (FID) field.

Origin Subarea Field (OSAF)
A field in an FID4 transmission header that contains a subarea address, which combined with the element address in the origin element field (OEF), gives the complete network address for the originating network addressable unit (NAU). Contrast with destination subarea field (DSAF). See also format identification (FID) field.

OSAF
See origin subarea field.

Pacing
A technique by which a receiving component controls the rate of transmission of a sending component to prevent overrun or congestion.

Pacing Group
1. The path information units (PIUs) that can be transmitted on a virtual route before a virtual route pacing response is received, indicating that the virtual route receiver is ready to accept more PIUs on the route.
2. The requests that can be transmitted on the normal flow in one direction in a session before a session level pacing response is received, indicating that the receiver is ready to accept the next group of requests.
3. Synonymous with window.

Pacing Response
An indicator that signifies a receiving component's readiness to accept another pacing group; the indicator is carried in a response header (RH) for session level pacing, and in a t See also isolated pacing response.

Parallel Sessions
Two or more concurrently active sessions between the same two logical units (LUs) using different pairs of network addresses. Each session can have independent session parameters.

Path
The series of path control network components (path control and data link control) that are traversed by the information exchanged between two network addressable units (NAUs). A path consists of a virtual route extension, if any. See also explicit route.

Path Control (PC) Layer
The layer that manages the sharing of link resources of the SNA network and routes basic information units (BIU) through it. Path control routes message units between network addressable units (NAUs) in the network and provides the paths between them. It converts the BIUs from transmission control (possibly segmenting them) into path information units (PIU) and exchanges basic transmission units (BTUs) – one or more PIUs – with data link control. See also BIU segment, data link control layer, transmission control (TC) layer.

NOTE
The unit of control information built by the sending path control component is the transmission header (TH), attached to the BTU; the TH is interpreted by the receiving path control component. The path control layer in subarea nodes consists of explicit route control, transmission group control, virtual route control, and boundary function path control.

NC-IPL-INIT
NC IPL initial.

NC-IPL-TEXT
NC IPL text.

Negative Response
A response indicating that a request did not arrive successfully or was not processed successfully by the receiver. Contrast with positive response. See also exception response.

Negotiable BIND
A capability that allows two LU-LU half sessions to negotiate the parameters of a session when the session is being activated.

Network Address
An address, consisting of subarea and element fields, that identifies a link, a link station, or a network addressable unit. Subarea nodes use network addresses; peripheral nodes use local addresses. The boundary function in the subarea node to which a peripheral node is attached transforms local addresses to network addresses and vice versa. See also local address.

Network Addressable Unit
A logical unit, a physical unit, or a system services control point. It is the origin or the destination of information transmitted by the path control network.

NOTE
Each NAU has a network address that represents it to the path control network (LUs can have multiple addresses for parallel LU-LU sessions). The path control network and the NAUs together constitute the SNA network.

Network Control (NC)
A request/response unit (RU) category used for requests and responses exchanged between physical units (PUs) for such purposes as activating and deactivating explicit and virtual routes and sending load modules to adjacent peripheral nodes. See also data flow control, function management data, session control.

Network Services
The services within network addressable units (NAUs) that control network operation via SSCP-SSCP, SSCP-PU, and SSCP-LU sessions. See also configuration services, maintenance services, management services, session services.

Network Services Header
A 3 byte field in an FMD request/response unit (RU) flowing in an SSCP-LU, SSCP-PU, or SSCP-SSCP session. The network services header is used primarily to identify the network services category of the RU (for example, configuration services, session services) and the particular request code within a category.

NMVT
Network management vector transport.

Node
An endpoint of a link or a junction common to two or more links in a network. Nodes can be distributed to host processors, communication controllers, or terminals. Nodes can vary in routing and other functional capabilities.

No Response
A value in the form of response requested field of the request header (RH) indicating that no response is to be returned to the request, whether or not the request is received and processed successfully. Contrast with definite response, exception response.

Normal Flow
A data flow designated in the transmission header (TH) that is used primarily to carry end user data. The rate at which requests flow on the normal flow can be regulated by session level pacing. Contrast with expedited flow.

NOTE
The normal and expedited flows move in both the primary to secondary and secondary to primary directions. Requests and responses on a given flow (normal or expedited) usually are processed sequentially within the path, but the expedited flow traffic can be moved ahead of the normal flow traffic within the path at queuing points in the half sessions and for half session support in boundary functions.

Notify
Used to synchronize awareness between SSCP and PU of status of a cross network session.

NS
See network services.

NS-IPL-ABORT
NS IPL abort.

NS-IPL-FINAL
NS IPL final.

RECTR
Record test results.

RECTRD
Record trace data.

REJ
SDLC reject frame.

RELQ
Release queues.

Reply
A request unit sent only in reaction to a received request unit. For example, Quiesce Complete is the reply sent after receipt of Quiesce At End of Chain.

REQACTLU
Request activate logical unit.

REQCONT
Request contact.

REQDISCONT
Request discontact.

REQECHO
Request echo test.

REQFNA
Request free network address.

REQMS
Request maintenance statistics.

Request
A message unit that signals initiation of a particular action or protocol. For example, INITIATE SELF is a request for activation of an LU–LU session.

Request Header (RH)
A request unit (RU) header preceding a request unit.

Request Unit (RU)
A message unit that contains control information such as a request code, or function management (FM) headers, end user data, or both.

Request/Response Header (RH)
Control information, preceding a request/response unit (RU), that specifies the type of RU (request unit or response unit) and contains control information associated with that RU.

Request/Response Unit (RU)
A generic term for a request unit or a response unit.

Response
1. A message unit that acknowledges receipt of a request; a response consists of a response header (RH), a response unit (RU), or both.
2. In SDLC, the control information (in the C–Field of the link header) sent from the secondary station to the primary station.

Response Header (RH)
A header, optionally followed by a response unit (RU), that indicates whether the response is positive or negative and that it can contain a pacing response.

Response Unit (RU)
A message unit that acknowledges a request unit; it can contain prefix information received in a request unit. If positive, the response unit can contain additional information (such as session parameters in response to BIND SESSION), or if negative, sense data defining the exception condition.

RH
See request/response header, request header, response header.

RIM
SDLC request initialization mode frame.

Route
See explicit route, virtual route.

RJE
Remote job entry.

RNAA
Request network address assignment.

RNR
SDLC receive not ready frame.

ROUTE–TEST
Route test.

RPO
Remote power off.

RQR
Request recovery.

RR
SDLC receive ready frame.

RSHUTD
Request shutdown.

Path Information Unit (PIU)
A message unit consisting of a transmission header (TH) alone, or of a TH followed by a basic information unit (BIU) or a BIU segment. See also transmission header.

PC
Path control.

PDI
Padded data indicator.

P/F
SDLC poll/final bit.

Physical Unit (PU)
The component that manages and monitors the resources (such as attached links and adjacent link stations) of a node, as requested by an SSCP via an SSCP–PU session. Each node of an SNA network contains a physical unit.

NOTE
An SSCP activates a session with the physical unit in order to indirectly manage, through the PU, resources of the node such as attached links and adjacent link stations.

PI
Pacing indicator. See also pacing response.

PIU
See path information unit.

Point To Point Line
A link that connects a single remote link station to a node; it can be switched or non–switched.

Positive Response
A response indicating that a request was successfully received and processed. Contrast with negative response.

Presentation Services (PS)
See session presentation services.

PRI
See primary link station.

Primary Half Session
The half session that sends the session activation request. Contrast with secondary half session.

Primary Link Station (PRI)
The link station on a link that is responsible for the control of that link. A link has only one primary link station. All traffic over the link is between the primary link station and a secondary link station. Contrast with secondary link station.

PROCSTAT
Procedure status.

Protocol
The meanings of, and the sequencing rules for, requests and responses used for managing the network, transferring data, and synchronizing the states of network components.

PS
See session presentation services.

PU
See physical unit.

Public Network
A network established and operated by communication common carriers or telecommunication administrations for the specific purpose or providing circuit switched, packet switched, and leased circuit services to the public.

PU–PU Flow
The exchange between physical units (PUs) of network control requests and responses.

QC
Quiesce complete.

QEC
Quiesce at end of chain.

RD
SDLC request disconnect frame.

Receive Pacing
The pacing of message units that a component is receiving. See also send pacing.

RECFMS
Record formatted maintenance statistics.

RECMS
Record maintenance statistics.

RECSTOR
Record storage.

RECTD
Record test data.

Session Activation Request
A request that activates a session between two network addressable units (NAUs) and specifies session parameters that control various protocols during session activity; for example, BIND and ACTPU. Contrast with session deactivation request.

Session Control (SC)
1. One of the components of transmission control. Session control is used to purge data flowing in a session after an unrecoverable error occurs, to resynchronize the data flow after such an error, and to perform cryptographic verification.
2. An RU category used for requests and responses exchanged between the session control components of a session and for session activation/deactivation requests and responses.

Session Count
1. The number of currently active LU–LU sessions for a particular logical unit.
2. The number of currently active sessions for a particular virtual route.

Session Deactivation
The process of exchanging a session deactivation request and response between network addressable units (NAUs). Contrast with session activation.

Session Deactivation Request
A request that deactivates a session between two network addressable units (NAUs); for example, UNBIND and DACTPU. Contrast with session activation request.

Session Initiation Request
An initiate or logon request from a logical unit (LU) to a system services control point (SSCP) that an LU–LU session be activated.

Session Limit
The maximum number of concurrently active LU–LU sessions a particular logical unit (LU) can support.

Session Level Pacing
A flow control technique that permits a receiving half session to control the data transfer rate (the rate at which it receives request units) on the normal flow. It is used to prevent overloading a receiver with unprocessed requests when the sender can generate requests faster than the receiver can process them. See also pacing, virtual route (VR) pacing.

Session Network Services
Network services that are performed on a half session by half session basis, rather than for the network addressable unit (NAU) as a whole.

Session Parameters
The parameters that specify or constrain the protocols (such as bracket protocol and pacing) for a session between two network addressable units (NAUs).

Session Partner
One of the two network addressable units (NAUs) having an active session.

Session Presentation Services (PS)
A component of the FMD services layer that provides, within LU–LU sessions, services for the application programmer or terminal operator such as formatting data to be displayed or printed.

Session Services
One of the types of network services in the system services control point (SSCP) and in a logical unit (LU). These services provide facilities for a logical unit (LU) or a network operator to request that the SSCP initiate or terminate sessions between logical units. See also configuration services, maintenance services, management services.

SESSST
Session started.

SETCV
Set control vector.

SHUTC
Shutdown complete.

SHUTD
Shutdown.

SIG
Signal.

SIM
SDLC set initialization mode frame.

SNA
Systems network architecture.

SNA Character String (SCS)
A data stream composed of EBCDIC controls, optionally intermixed with end user data, that is carried within a request/response unit.

RTI
Response type indicator.

RTR
Ready to receive.

RU
See request/response unit, request unit, response unit.

RU Chain
A set of related request/response units (RUs) that are consecutively transmitted on a particular normal or expedited data flow. The request RU chain is the unit of recovery: if one of the RUs in the chain cannot be processed, the entire chain is discarded.

NOTE
Each RU belongs to only one chain, which has a beginning and an end indicated via control bits in request/response headers within the RU chain. Each RU can be designated as first-in-chain (FIC), last-in-chain (LIC), middle-in-chain (MIC), or only-in-chain (OIC). Response units and expedited flow request units are always sent as only-in-chain.

SBI
Stop bracket initiation.

SC
See session control.

SCS
See SNA character string.

SDI
Sense data included indicator.

SDLC
Synchronous data link control.

SDT
Start data traffic.

SEC
See secondary link station.

Secondary Half Session
The half session that receives the session activation request. Contrast with primary half session.

Secondary Link Station
Any link station on a link, using a primary-secondary protocol, that is not the primary link station. A secondary link station can exchange data only with the primary link station; no data traffic flows from one secondary link station to another. Contrast with primary link station.

Secondary Station
See secondary link station.

Segmenting of BIUs
An optional function of path control that divides a basic information unit (BIU) received from transmission control into two or more path information units (PIUs). The first PIU contains the request header (RH) of the BIU and usually part of the RU; the remaining PIU or PIUs contain the remaining parts of the RU.

NOTE
When segmenting is not done, a PIU contains a complete BIU.

Send Pacing
Pacing of message units that a component is sending. See also receive pacing.

SESSEND
Session end.

Session
A logical connection between two network addressable units (NAUs) that can be activated, tailored to provide various protocols, and deactivated, as requested. The session activation request and response can determine options relating to such things as the rate and concurrency of data exchange, the control of contention and error recovery, and the characteristics of the data stream. Sessions compete for network resources such as the links within the path control network. See also half session, LU-LU session, SSCP-LU session, SSCP-PU session, SSCP-SSCP session.

NOTE
For routing purposes, each session is identified by the network (or local) address of the session partners.

Session Activation
The process of exchanging a session activation request and a positive response between network addressable units (NAUs). Contrast with session deactivation.

TC
See transmission control layer.

TERM–OTHER
Terminate other.

TERM–OTHER–CD
Terminate other cross–domain.

TERM–SELF
Terminate self.

Terminal Node
A peripheral node that is not user–programmable, having less processing capability than a cluster controller node. Examples are the IBM 3277, 3767, 3614, and 3624.

TESTMODE
Test mode.

TH
See transmission header.

Transmission Control (TC) Layer
The layer within a half session that synchronizes and paces session level data traffic, checks session sequence numbers of requests, and enciphers and deciphers end user data. Transmission control has two components: the connection point manager and session control.

Transmission Header (TH)
Control information, optionally followed by a basic information unit (BIU) or a BIU segment, that is created and used by path control to route message units and to control their flow within the network. See also path information units.

UA
SDLC unnumbered acknowledgment frame.

UI
SDLC unnumbered information frame.

UNBIND
Unbind session.

UNBINDF
Unbind failure.

UP
SDLC unnumbered poll.

User Application Network
A configuration of data processing products (such as processors, controllers, and terminals) established and operated by users for the purpose of data processing or information exchange, which can use services offered by common carriers or telecommunication administrations. Contrast with public network.

Virtual Route (VR)
A logical connection:

1. Between two subarea nodes that is physically realized as a particular explicit route.
2. That is contained wholly within a subarea node for intranode sessions. A virtual route between distinct subarea nodes imposes a transmission priority on the underlying explicit route, provides flow control through virtual route pacing. and provides data integrity through sequence numbering of path information units (PIUs). See also explicit route (ER), path.

Virtual Route (VR) Pacing
A flow control technique used by the virtual route control component of path control at each end of a virtual route to control the rate at which path information units (PIUs) flow over the virtual route. VR pacing can be adjusted according to traffic congestion in any of the nodes along the route. See also pacing, session level pacing.

VR
See virtual route.

VR–INOP
Virtual route inoperative.

Window
Synonym for pacing group.

Window Size
Synonym for pacing group size.

XID
SDLC exchange identification frame.

SNA Network
The part of a user application network that conforms to the formats and protocols of Systems Network Architecture. It enables reliable transfer of data among end users and provides protocols for controlling the resources of various network configurations. The SNA network consists of network addressable units (NAUs), boundary function components, and the path control network.

SNA Node
A node that supports SNA protocols.

SNA Station
A station that supports SNA protocols.

SNA Terminal
A terminal that supports SNA protocols.

SNF
Sequence number field.

SSCP
See system services control point.

SSCP ID
A number that uniquely identifies a system services control point (SSCP). The SSCP ID is used in session activation requests sent to physical units (PUs) and to other SSCPs.

SSCP–LU Session
A session between a system services control point (SSCP) and a logical unit (LU); the session enables the LU to request the SSCP to help initiate LU–LU sessions.

SSCP–PU Session
A session between a system services control point (SSCP) and a physical unit (PU); SSCP–PU sessions allow SSCPs to send requests to and receive status information from individual nodes in order to control the network configuration.

SSCP Services
The components within a system services control point (SSCP) that provide configuration, maintenance, management, network, and session services for SSCP–LU, SSCP–PU and SSCP–SSCP sessions.

SSCP Services Manager
An SNA component that provides network services for all the half sessions of the system services control point (SSCP).

SSCP–SSCP Session
A session between the system services control point (SSCP) in one domain and the SSCP in another domain. An SSCP–SSCP session is used to initiate and terminate cross domain LU–LU sessions.

Station
1. A link station.
2. One or more computers, terminals, devices, and associated programs at a particular location.

STSN
Set test and test sequence numbers.

Synchronous Data Link Control (SDLC)
A discipline for managing synchronous, code–transparent, serial by bit information transfer over a link connection. Transmission exchanges can be full duplex or half duplex over switched or nonswitched links. The configuration of the link connection can be point to point, multipoint, or loop. SDLC conforms to subsets of the Advanced Data Communication Control Procedures (ADCCP) of the American National Standards Institute and High–Level Data Link Control (HDLC) of the International Standards Organization.

System Services Control Point (SSCP)
A focal point within an SNA network for managing the configuration, coordinating network operator and problem determination requests, and providing directory support and other session services for end users of the network. Multiple SSCPs, cooperating as peers with one another, can divide the network into domains of control, with each SSCP having a hierarchical control relationship to the physical units and logical units within its own domain.

Systems Network Architecture (SNA)
The description of the logical structure, formats, protocols, and operational sequences for transmitting information units through and controlling the configuration and operation of networks.

NOTE
The purpose of the layered structure of SNA is to allow the ultimate origins and destinations of information – that is, the end users – to be independent of, and unaffected by, the way in which the specific SNA network services and facilities used for information exchange are provided.

Frame Identifiers					
Frame Type	R*RR	R*RNR	R*REJ	R*REJ	R*UI
	R*SNRM	R*DISC	R*SIM	R*SIM	R*TEST
	R*XID	R*UA	R*RD	R*RIM	R*DM
	R*UP	R*FRMR	R*I		

Table D–3 Frame Identifiers

Frame Events		
Command	**Stack Description**	**Description**
?RX_BLU	(frame identifier -- flag)	Detects a frame or BLU
?RX_FRAME	(id#1\--\id#n\n--flag) Where n = number of identifiers	Detects for one of a group of frames or BLUs
?ADDRESS	(PU address -- flag)	Detects a frame with a specific address
FRAME?	(--)	Detects any type of frame
DTE?	(--)	Detects any type of frame from the DTE
DCE?	(--)	Detects any type of frame from the DCE
?MATCH_I	(" string" -- flag)	Detects a specific string in an I frame
?SEARCH_I	(" string" -- flag)	Detects a specific string anywhere in an I frame

Table D–4 Frame Events

D
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 D–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, V.35, V.36)
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 D–2 Setting Leads

SNA Header Events		
Command	**Stack Description**	**Description**
?THAF	(address field -- flag)	Detects a specific origination or destination address field
?MAPPING_FIELD	(mapping field -- flag)	Detects a specific mapping field
?RX_REQ	(-- flag)	Detects a request type response/ request header
?RX_RES	(-- flag)	Detects a response type response/ request header
?RX_RH	(RU category -- flag)	Detects a specific request/response header
?RX_RU	(RU value -- flag)	Detects a specific response unit header

Table D-6 SNA Header Events

Sending Frames and BTUs		
Command	**Stack Description**	**Description**
SENDF	("string" --)	Sends a string as a frame
SEND_BTU	("string" --)	Sends a string as a BTU
SEND_BUFFER	(buffer number --)	Sends a buffer as an I frame
SNRM , S:SNRM	(--)	Sends a SNRM frame
DISC, S:DISC	(--)	Sends a DISC frame
SIM, S:SIM	(--)	Sends a SIM frame
XID, S:XID	(--)	Sends a XID frame
RIM, S:RIM	(--)	Sends a RIM frame
RM, S:RM	(--)	Sends a RM frame
S:UA	(--)	Sends a UA frame
S:DM	(--)	Sends a DM frame
S:UP	(--)	Sends a UP frame
S:RR	(--)	Sends a RR frame
S:RNR	(--)	Sends a RNR frame
S:REJ	(--)	Sends a reject frame
BUILD_TH_BYTE0		Builds byte 0 of the TH to send using SEND_BUFFER
BUILD_TH	(string -- string)	Builds the TH to send using SEND_BTU
BUILD_RH	(string -- string)	Builds RH to send using SEND_BTU
BUILD_RU	(string -- string)	Builds RU to send using SEND_BTU

Table D-7 Sending Frames and BTU's

SNA Header Identifiers				
TH constants	R*FID0	R*FID1	R*FID2	R*FID3
	R*FID4	R*FIDF	R*INVFID	
Mapping field constants	R*MIDDLE-SEGMENT	R*FIRST-SEGMENT	R*LAST-SEGMENT	
	R*WHOLE-BIU			
RH constants	R*FMD	R*DFC	R*NC	R*SC
RU constants (SC) category	R*ACTDRM	R*ACTLU	R*ACTPU	R*BIND
	R*CLEAR	R*CRV	R*DACTCDRM	R*DACTLU
	R*DACTPU	R*RQR	R*SDT	R*STSN
	R*UNBIND			
RU constants (DFC) category	R*BID	R*BIS	R*CANCEL	R*CHASE
	R*LUSTAT	R*QC	R*QEC	R*RELQ
	R*RSHUTD	R*RTR	R*SBI	R*SHUTC
	R*SHUTD	R*SIG		
RU constants (NC) category	R*ANSC	R*LSA	R*NC-ACTVR	R*NC-DACTVR
	R*NC-ER-ACT	R*NC-ER-TEST-REPLY	R*NC-ER-INOP	R*NC-ER-OP
	R*NC-ER-TEST	R*NC-IPL-FINAL	R*NC-ER-ACT-REPLY	R*NC-IPL-INIT
	R*NC-IPL-TEXT			
RU constants (FMD) category	R*ABCONN	R*ABCONNOUT	R*ACTCONNIN	R*ACTLINK
	R*ACTTRACE	R*ADDLINK	R*ADDLINKSTA	R*ANA
	R*BINDF	R*CDCINIT	R*CINIT	R*CDSESSEND
	R*CDSESSSF	R*CDSESSST	R*CDSESSTF	R*CDTAKED
	R*CDTAKEDC	R*CDTERM	R*CESLOW	R*CEXLOW
	R*CLEANUP	R*CONNOUT	R*CONTACT	R*CONTACTED
	R*CTERM	R*DACTCONNIN	R*DACTLINK	R*DACTTRACE
	R*DELETENR	R*DELIVER	R*DISCONTACT	R*DISPSTOR
	R*DSRLST	R*DUMPFINAL	R*DUMPINIT	R*DUMPTEXT
	R*ECHOTEST	R*ER-INOP	R*ER-TESTED	R*ESLOW
	R*EXECTEST	R*EXSLOW	R*FNA	R*FORWARD
	R*INIT-OTHER	R*INIT-OTHER-CD	R*INIT-SELF_F0	R*INIT-SELF_F1
	R*INITPROC	R*INOP	R*IPLFINAL	R*IPLINIT
	R*IPLTEXT	R*ISETCV	R*LCP	R*LDREQD
	R*NMVT	R*NOTIFY_SSCP-LU	R*NS-IPL-ABORT	R*NS-IPL-FINAL
	R*NS-IPL-INIT	R*NS-IPL-TEXT	R*NS-LSA	R*NSLSA
	R*NSPE	R*PROCSTAT	R*RECFMS	R*RECMD
	R*RECMS	R*RECSTOR	R*RECTD	R*RECTR
	R*RECTRD	R*REQACTLU	R*REQCONT	R*REQDISCONT
	R*REQECHO	R*REQFNA	R*REQMS	R*REQTEST
	R*RNAA	R*ROUTE-TEST	R*RPO	R*SESSEND
	R*SESSST	R*SETCV-CONFIG	R*SETCV-MAINT	R*STARTMEAS
	R*TERM-OTHER	R*TERM-OTHER-CD	R*TERM-SELF	R*TERM-SELF_F1
	R*TESTMODE	R*UNBINDF	R*VR-INOP	

Table D-5 SNA Header Identifiers

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 D–11 Creating User Output

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 D-8 Creating Buffers

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 D-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 D-10 Timer Events

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 D-12 Program Control Events

E.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 E-1 ITL Symbols

E.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.

E
CODING CONVENTIONS

The following 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.

E.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 left–most item representing the bottom of the stack and the right–most 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 or a 1 is returned on the stack after the '=' operation, with 0 being a false result, and 1 a true result.

E.6 Colon Definitions

Colon definition should be preceded by a short comment. The colon definition should start at the first column of a line. All code 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
;
```

E.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. Furthermore, the use of many meaningful comments is highly recommended and will 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
```

E.5 Spacing and Indentation Guidelines

The following 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.);
- tab for nested constructs;
- carriage return after colon definition and stack comment; and
- carriage return after last line of code in colon definition and semi-colon;

See the examples in Appendices E.4 and E.6.

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		†

HEX	DEC	OCT	ASCII	EBCDIC
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		–

F
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
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		

G
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
CFRMA=	DO_CFRMA
CTHAF=	DO_CTHAF
DATA-STATUS	STATUS_ERR?
DO_SETID	SET_LINK_ID
DFRMA=	DO_DFRMA
DTHAF=	DO_DTHAF
PLAY-ID	PORT-ID
PLAY-STIME	START-TIME
PLAY-ETIME	END-TIME
PLAY-COUNT	BLOCK-COUNT
POLL_TIME=	DO_PTIME
REC-STATUS/DATA_STATUS	STATUS_ERR?
RFRMA=	DO_RFRMA
RTHAF=	DO_RTHAF
SET_#LINKS=	DO_#LINKS
SET_ID=	SET_LINK_ID
SET_LINK#=	DO_LINK#
SET_LONG	LONG-INTERVAL
SET_RETRY#=	DO_SETRETRY
SET_SPEED	INTERFACE-SPEED
SET_SHORT	SHORT-INTERVAL
SET_WINDOW=	DO_SETW

INDEX [continued]

INDEX

INDEX [continued]

INDEX [continued]

INDEX [continued]

INDEX [continued]

INDEX [continued]