EY-8324E-SL-0001

DECnet-VAX Internals

Supplemental Listings

Prepared by Educational Services
of
Digital Equipment Corporation

EY-8324E-SL-0001

First Edition, July 1987

Printed in U.S.A.

The manuscript for this book was created using DIGITAL Standard Runoff. Book production was done by Educational Services Development and Publishing in Nashua, NH.

CONTENTS

1 DDCMP MESSAGE FORMATS

2 ROUTER LAYER MESSAGE FORMATS

3 NSP (ECL) MESSAGE FORMATS

4 SESSION CONTROL LAYER MESSAGES

5 NETWORK INFORMATION AND CONTROL EXCHANGE PROTOCOL

6 DECnet-VAX SDL FILES

DDCMP MESSAGE FORMATS

INTRODUCTION

This section describes the message formats of DDCMP. Data is exchanged over DDCMP links between the data source (master) and data sink (slave) within numbered data messages. Responses and control information are returned from the slave to the master within unnumbered control messages. Stations contain both a master and slave. For the purpose of exchanging data, the station plays the role of master or slave depending on whether it is transmitting or receiving the data. It is a distinction used for easy understanding and explanation of DDCMP. In reality, data is usually exchanged in both directions. In the following explanation only a single direction is described.

Each data message carries a number assuring correct message sequencing at the slave. The numbering begins with number one after initialization via the STRT/STACK control message sequence and is incremented by one (modulo 256) for each subsequent data message. The slave always acknowledges the correct receipt of data messages by returning the message number as a response either in the response field of numbered data messages being sent or, in an ACK unnumbered control message. For efficiency, an acknowledgement of the data message with number n implies an acknowledgement of all consecutive data messages sent up to and including data message number n. Retransmission is used to recover from errors. The error recovery mechanism uses timeouts and NAK and REP control messages to resynchronize and cause retransmission if required. All messages also include station addresses and link control flags for use on multipoint and half-duplex channels.

1 DNA MESSAGE FORMAT NOTATION

The following notation is used to describe all DNA messages:

FIELD (LENGTH) : CODING Description of field

FIELD is the name of the field.

(LENGTH) is the length of the field, one of:

1. A number meaning the number of 8-bit bytes.
2. A number followed by a "B" meaning the number of bits.
3. The letters "I-n" meaning an image field, with n being a number that specifies the maximum length of 8-bit bytes in the image. The image is preceded by a 1-byte count of the length of the remainder of the field. Image fields are of variable length and may be null (count = 0). All eight bits of each byte are information bits.

CODING represents the type of coding used, one of:

1. B = Binary.
2. BM = Bit map. Each bit has independent meaning.
3. C = Constant.
4. NULL = Interpretation is data-dependent.

1.1 Notes on DNA Notation

1. Fields in separate messages with identical names are the same field and have identical meanings.

2. All numeric values are decimal unless otherwise noted.

3. All header fields and data bytes are transmitted low-order or least-significant-bit first on the data line unless otherwise noted.

4. Multiple byte fields are transmitted low-order or least-significant-byte first.

5. Bits are numbered with bit 0 on the right (low-order, least-significant bit) and bit 7 on the left (high-order, most-significant bit). For convenience, when the graphic form of a 2-byte field is given, it will be shown converted to a 16-bit word. When a subfield of a message field contains more than one bit, it should be considered a binary value.

6. Fields in separate messages that have the identical name are the same field and have identical meaning.

7. All numeric values in this document are shown in decimal representation unless otherwise noted.

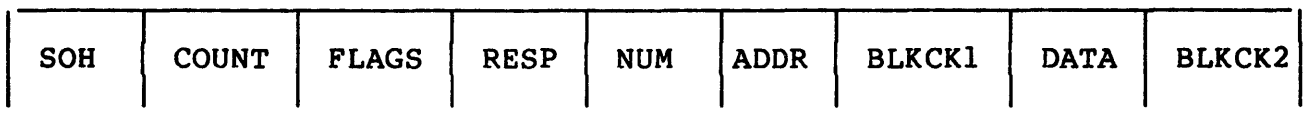
8. Unless otherwise specified, the numbers that appear at the top of the message formats represent bit positions.

9. Bracketed fields are optional.

2 DATA MESSAGES

Numbered data messages carry user data over DDCMP links.

The format of a numbered message is:



SOH(1) : C = The numbered data message identifier. It has a value of 129 (octal - 201, hex - 81).

COUNT(14B) : B = The byte count field. It specifies the number of 8-bit bytes in the DATA field. The value zero is not allowed.

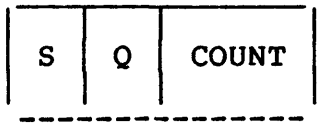
FLAGS(2B) : BM = The link flags. They are used to control link ownership and message synchronization. These flags are:

bit 0 = Quick sync flag (QSYNC flag), used to notify the receiver that the next message will not abut this message and resynchronization should follow this message. The quick sync flag reduces the length of sync sequences on synchronous links.

bit 1 = Select flag (SELECT flag), used to control transmission ownership on multipoint and half-duplex links. Reverses link direction on half-duplex links. Invites a tributary to send and signals end of tributary selection on multipoint links.

NOTE

COUNT and FLAGS form a 2-byte quantity. The first byte contains the 8 low-order bits of the COUNT. The second byte contains the 6 high-order bits of the COUNT, the SELECT flag, the highest order or most-significant-bit of the byte, and the QSYNC flag, the next bit in the byte.



High-order bit transmitted last

Low-order bit transmitted first

RESP(1) : B = The response number. It is used to acknowledge correctly received messages (the piggybacked ACK). It is the number of the last consecutive correctly received message received from the addressed station by the station transmitting this message. It implies that all unacknowledged messages between the one acknowledged in the last RESP field received and the one acknowledged by this RESP field (modulo 256), have been received correctly.

NUM(1) : B = The transmit number. It is used to denote the number of this data message.

ADDR(1) : B = The station address field. It is used to designate the address of tributary stations on multipoint links. Stations on point-to-point links use the address value 1.

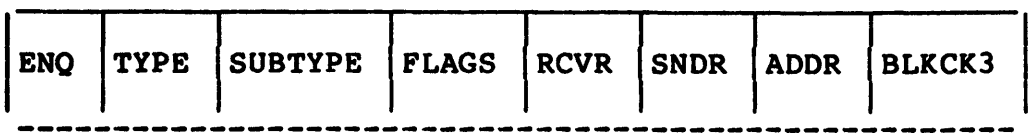
BLKCK1(2) : B = The block check on the numbered message header. It is computed on SOH through ADDR using the CRC-16 polynomial (X^16+X^15+X^2+1). BLKCK1 is initialized to zero prior to computation and transmitted X^15 bit first. On reception the inclusion of BLKCK1 in the computation will result in a zero remainder or CRC value if no bit errors exist.

DATA (COUNT) = The numbered message data field. This field is totally transparent to the protocol and has no restrictions on bit patterns, groupings, or interpretations. The only requirement is that it contain the number of 8-bit bytes specified in the COUNT field.

BLKCK(2) : B = The block check on the data field. It is computed on the DATA field only using the polynomial and technique described above for BLKCK1.

3 CONTROL MESSAGES

Unnumbered control messages carry channel information, transmission status, and initialization notification between the protocol modules. The individual fields are specific for each type of control message. Control messages have the following general form:



ENQ(1) : C = The unnumbered control message identifier. It has a value of 5 (octal - 005, hex - 05).

TYPE(1) : B = The control message type. This value denotes the type of each control message.

SUBTYPE(6B) : B = The subtype or type modifier field. It provides additional information for some message types. Its use is specific for each message type.

FLAGS(2B) : BM = The link flags. They are the same as described for numbered data messages.

RCVR(1) : B = The control message receiver field. It is used to pass information from the data message receiver or slave station to the data message sender or master station. Its use is specific for each control message type.

SNDR(1) : B = The control message sender field. It is used to pass information from the data message sender or master to the data message receiver or slave. Its use is specific for each control message type.

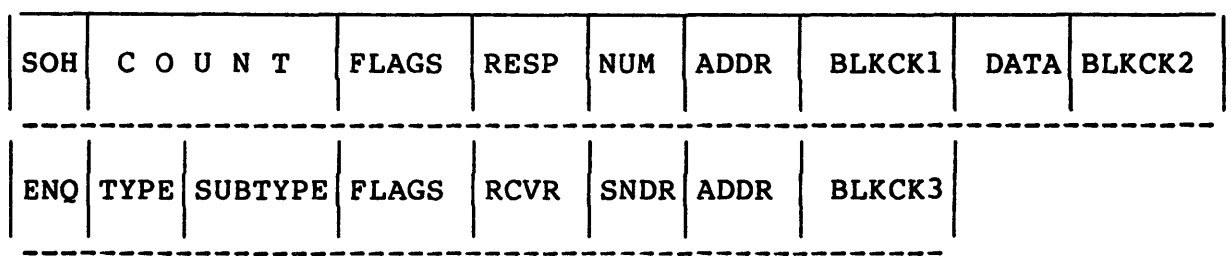
ADDR(1) : B = The station address field. It is the same as described for numbered data messages.

BLKCK3(2) : B = The block check on the control message. BLKCK3 is computed on fields ENQ through ADDR using the polynomial and technique described for numbered data message BLKCK1

NOTE

The common fields in data and control messages are in the same position relative to the beginning of the message.

The two types line up as follows:



4 ACKNOWLEDGE MESSAGE (ACK)

The ACK message is used to acknowledge the correct receipt of numbered data messages. It conveys the same information as the RESP field in numbered messages and is used when acknowledgements are required, and when no numbered messages are to be sent in the reverse direction. The form of the ACK message is:

ENQ(1) : C = The control message identifier.

ACKTYPE(1) : C = The ACK message type with a value of 1.

ACKSUB(6B) : C = The ACK subtype with a value of 0.

FLAGS(2B) : BM = The link flags.

RESP(1) : B = The response number used to acknowledge correctly received messages. It is the same as described for numbered data messages.

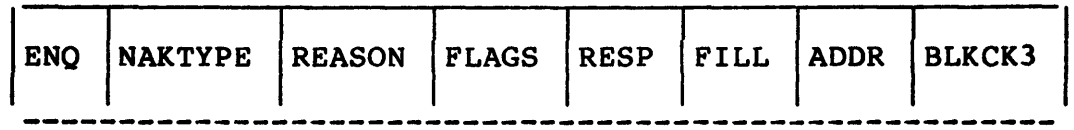
FILL(1) : C = A fill byte with value 0.

ADDR(1) : B = The station address field.

BLKCK3(2) : B = The control message block check.

5 NEGATIVE ACKNOWLEDGE MESSAGE (NAK)

The NAK message is used to pass error information from the slave (or data receiver) to the master (or data sender). The error reason is included in the subtype field. The NAK message also includes the same information as the ACK message, thus serving two functions: acknowledging previously received messages and notifying the master of some error condition. The form of the NAK message is:



ENQ(1) : C = The control message identifier.

NAKTYPE(1) : C = The NAK message type with a value of 2.

REASON(6B) : B = The NAK error reason. Identifies the source and reason for the NAK.

1. Error usually due to transmission medium:
   Value and Reason
   1 = header block check error (data message BLKCK1 or control message BLKCK3).
   2 = data field block check error (data message BLKCK2).
   3 = REP response.

2. Error usually due to computer and/or interface:
   Value and Reason
   8 = buffer temporarily unavailable.
   9 = receiver overrun.
   16 = message too long.
   17 = message header format error.

FLAGS(2B) : BM = The link flags.

RESP(1) : B = The response number used to acknowledge correctly received messages. When used in a NAK message, usually implies some error in a message with number RESP+1 (modulo 256) or beyond.

FILL(1) : C = A fill byte with a value of 0.

ADDR(1) : B = The station address field.

BLKCK3(2) : B = The control message block check.

6 REPLY TO MESSAGE NUMBER (REP)

The REP message is used to request received message status from the slave or data receiver. It is usually sent when the master has transmitted data messages and has not received a reply within a time-out period. The response to a REP is either an ACK or NAK depending on whether the slave has or has not received all messages previously sent by the master. The form of the REP message is:

ENQ(1) : C = The control message identifier.

REPTYPE(1) : C = The REP message type with a value of 3.

REPSUB(6B) : C = The REP subtype with a value of 0.

FLAGS(2B) : BM = The link flags.

FILL(1) : C = A fill byte with a value of 0.

NUM(1) : B = The number of the last sequential numbered data message (not including retransmissions) sent by the master. This is compared against the number of the last sequential message received by the slave and results in either an ACK being returned if they agree or a NAK if they do not. The NAK will contain the number of the last sequential message that was received.

ADDR(1) : B = The station address field.

BLKCK3(2) : B = The control message block check.

7 START MESSAGE (STRT)

The STRT message is used to establish initial contact and synchronization on a DDCMP link. It is used only on link startup or reinitialization. It operates with the start acknowledge message STACK described below. The start sequence resets message numbering at the transmitter and addressed receiver. The form of the STRT message is:

ENQ(1) : C = The control message identifier.

STRTTYPE(1) : C = The STRT message type with a value of 6.

STRTSUB(6B) : C = The STRT subtype with a value of 0.

FLAGS(2B) : C = The link flags. For STRT, both flags are ones (flags value of 3).

FILL(1) : C = A fill byte with a value of 0.

FILL(1) : C = A fill byte with a value of 0.

ADDR(1) : B = The station address field.

BLKCK3(2) : B = The control message block check.

8 START ACKNOWLEDGE MESSAGE (STACK)

The STACK message is returned in response to a STRT when the station has completed initialization and has reset its message numbering. The form of the STACK message is:

ENQ(1) : C = The control message identifier.

STCKTYPE(1) : C = The STACK message type with a value of 7.

STCKSUB(6B) : C = The STACK subtype with a value of 0.

FLAGS(2B) : C = The link flags. For STACK, both flags are ones (flags value of 3).

FILL(1) : C = A fill byte with a value of 0.

FILL(1) : C = A fill byte with a value of 0.

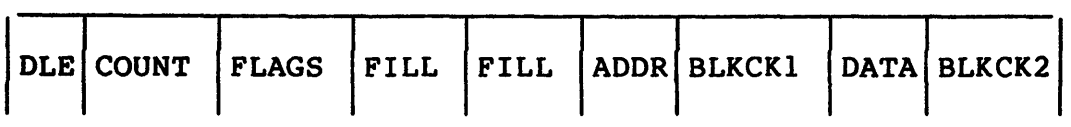
ADDR(1) : B = The station address field.

BLKCK3(2) : B = The control message block check.

9 MAINTENANCE MESSAGES

The DDCMP protocol operates in two basic modes: a) on-line, or the normal running mode and b) off-line, or the maintenance mode.

The previous messages and operation describe the on-line mode. The off-line, or maintenance mode, may be used for basic diagnostic testing and simple operating procedures such as bootstrapping, down-line loading, or dumping. It provides a basic envelope compatible with DDCMP framing, link management, and the CRC check for bit errors, but does not include any error recovery, retransmission timeouts, or sequence checks. All these functions, if necessary, are handled by the user of this mode within the data field. The maintenance message is similar in format to the data message. The format of the maintenance message is:



DLE(1) : C = The maintenance message identifier has the value 144 (octal - 220, hex - 90).

COUNT(14B) : B = The byte count field specifies the number of 8-bit bytes in the DATA field. The value zero is not allowed.

FLAGS(2B) : C = The link flags. Both flags are ones for maintenance messages (flags value of 3).

FILL(1) : C = A fill byte with a value of 0.

FILL(1) : C = A fill byte with a value of 0.

ADDR(1) : B = The station address field.

BLKCK1(2) : B = The header block check on fields DLE through ADDR. (Same as described for data messages)

DATA (COUNT) = The data field. It consists of COUNT 8-bit bytes.

BLKCK2(2) : B = The block check on the DATA field only. (Same as described for numbered data messages)

ROUTER LAYER MESSAGE FORMATS

INTRODUCTION

This section describes the message formats of the Routing Layer protocol. There are two types of Routing Layer messages:

- Packet route header -- This is used for ECL segments, which may require forwarding. There are two possible formats for data packet route headers:

  1. Short format (identical to Phase III format)

  2. Long format

- Routing Layer control -- These control Routing Layer routing and initialization functions. On non-broadcast circuits the types of Routing Layer control messages are:

  1. Initialization Message

  2. Verification Message

  3. Hello and Test Message

  4. Level 1 Routing Message

  5. Level 2 Routing Message

  On broadcast circuits the types of Routing Layer control messages are:

  1. Ethernet Router Hello Message

  2. Ethernet Endnode Hello Message

  3. Level 1 Routing Message

  4. Level 2 Routing Message

1 RESERVED FIELDS

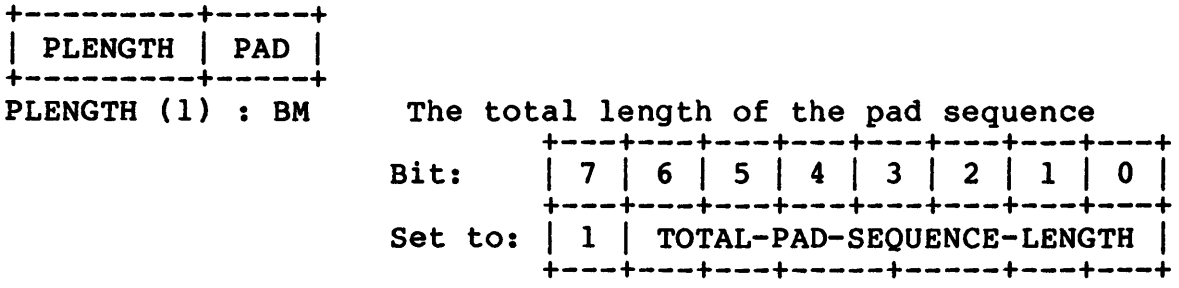
Reserved fields in all received packets are ignored and transmitted as 0, except that reserved bits set in received data packets to be forwarded are passed along unchanged. If translating between long and short format, a reserved bit which was set in a field to be dropped is dropped along with the field to be dropped.

2 OPTIONAL PADDING

All Routing Layer messages except Initialization can be padded. Padding can be used when communicating with Phase IV nodes, but must not be used when communicating with adjacent Phase III nodes.

If the top bit of the first byte is set, the remainder of the first byte is a count of the number of pad bytes, including the first byte. The total length of a message, including the padding, must not exceed the neighbor's blocksize. If the neighbor's blocksize is unknown (as in the case of Ethernet endnodes), then the maximum total pad sequence length is 7.

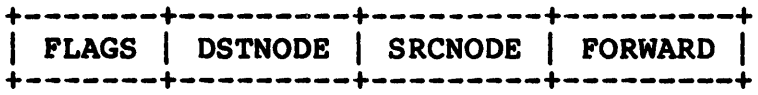
Thus, the format of the optional padding is as follows:



PAD ((TOTAL-PAD-SEQUENCE-LENGTH) - 1) : No Meaning

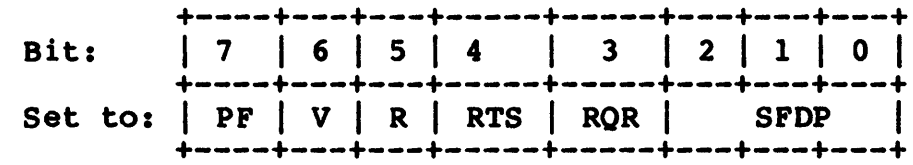
3 SHORT DATA PACKET FORMAT

The packet route header in short format is as follows:



FLAGS (1) : BM The set of flags used by the routing nodes.

The format of this field is as follows:

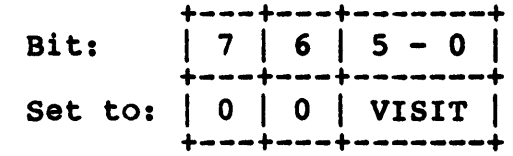


Bit	Definition
0-2	SFPD = 2, meaning short format
3	RQR (Return to Sender Request) 1 indicates try to return 0 indicates discard
4	RTS Return to Sender 1 => packet is on return trip
5	Reserved
6	Version, set to 0
7	PF pad field = 0 indicating no padding follows

DSTNODE (2) : B The destination node address

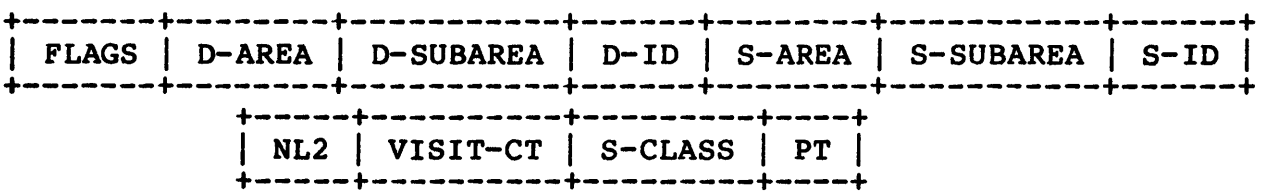
SRCNODE (2) : B The source node address

FORWARD (1) : BM Information useful in the forwarding of the message. The format of this field is as follows:



VISIT (6B) : BM the count of the number of nodes visited by this packet

4 LONG DATA PACKET FORMAT



FLAGS (1) : BM The set of flags used by the routing nodes.

The format of this field is as follows:

Bit:	7	6	5	4	3	2	1	0
Set to:	PF	V	I-E	RTS	RQR	LFDP		

Bit	Definition
0-2	LFDP = 6, meaning long format
3	RQR (Return to Sender Request) 1 indicates try to return 0 indicates discard
4	RTS Return to Sender 1 => packet is on return trip
5	Intra-Ethernet packet
6	Version, set to 0 discarded if 1
7	PF pad field = 0 indicating no padding follows

D-AREA (1) : B Reserved

D-SUBAREA (1) : B Reserved

D-ID (6) : B Destination ID (first 4 bytes must be set to HIORD)

S-AREA (1) : B Reserved

S-SUBAREA (1) : B Reserved

S-ID (6) : B Source ID (first 4 bytes must be set to HIORD)

NL2 (1) : B	Next level 2 router, reserved
VISIT-CT (1) : B	Visit Count (0)
S-CLASS (1) : BM	Service Class, reserved
PT (1) : B	Protocol Type, reserved

5 INITIALIZATION MESSAGE

FLAGS	SRCNODE	TIINFO	BLKSIZE	TIVER	TIMER	RESERVED

FLAGS (1) : BM The Routing Layer control flag, with the following format:

Bit:	7	6	5	4	3	2	1	0
Set to:	PF	RES			TYPE			1

Bit	Definition
0	1 indicates Control Packet
1-3	Type = 0
4-6	Reserved
7	PF pad field = 0 indicating no padding follows

SRCNODE (2) : B The identification of the source node's Routing Layer, containing the value ID as set by Network Management in the SELF parameters.

TIINFO (1) : BM Routing Layer information on node type and service requests, as follows:

Bit:	7	6	5	4	3	2	1	0
Set to:	0	0	0	0	BLO	VERIF	NTYPE	

NTYPE (2B) : B the Routing Layer node type:

0 Reserved
1 level 2 router
2 level 1 router
3 endnode

VERIF (1B) : BM Routing Layer Verification Message required if this bit is set.

BLO (1B) : BM Blocking Requested if this bit is set.

BLKSIZE (2) :	B	The maximum Data Link Layer receive block size this node will accept. (includes Routing header, excludes Data Link header)
TIVER (3) :	B	The Routing Layer version, with the following format: Byte 1 -- version number (2 (00000010 binary)) Byte 2 -- ECO number (0 (00000000 binary)) Byte 3 -- user ECO number (0 (00000000 binary))
TIMER (2) :	B	Hello Timer, in seconds
RESERVED (I-64)		A reserved field containing a count of 0.

6 VERIFICATION MESSAGE

```
+-------+---------+---------+
| FLAGS | SRCNODE | FCNVAL |
+-------+---------+---------+
```

FLAGS (1) : BM The Routing Layer control flag, with the following format:

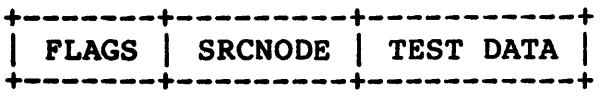
```
          +---+---+---+---+---+---+---+---+
Bit:      | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
          +---+---+---+---+---+---+---+---+
Set to:   |PF |    RES    |    TYPE   | 1 |
          +---+---+---+---+-----------+---+
```

Bit	Definition
0	1 indicates Control Packet
1-3	Type = 1
4-6	Reserved
7	PF pad field = 0 indicating no padding follows

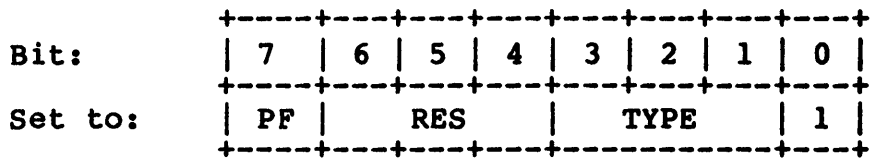
SRCNODE (2) : B The identification of the source node's Routing Layer, containing the value ID as set by Network Management in the SELF parameters.

FCNVAL (I-64) : B The function value.

7 HELLO AND TEST MESSAGE



FLAGS (1) : BM The Routing Layer control flag, with the following format:



Bit	Definition
0	1 indicates Control Packet
1-3	Type = 2
4-6	Reserved
7	PF pad field = 0 indicating no padding follows

SRCNODE (2) : B The identification of the source node's Routing Layer, containing the value ID as set by Network Management in the SELF parameters.

TEST DATA (I-128) : B A sequence of 0 to 128 bytes of data used to test the circuit. Each byte is 252 octal.

8 LEVEL 1 ROUTING MESSAGE

A Level 1 Routing Message contains one or more segments, each segment referring to COUNT destinations starting with STARTID.

Destination 0 is understood to mean "nearest level 2 router".

originator

FLAGS	SRCNODE	RES	SEGMENT	SEGMENT	...	CHECKSUM

FLAGS (1) : BM The Routing Layer control flag, with the following format:

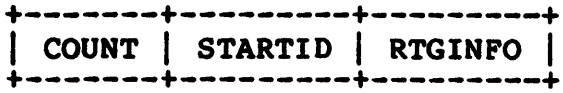
Bit:	7	6	5	4	3	2	1	0
Set to:	PF	RES			TYPE			1

Bit	Definition
0	1 indicates Control Packet
1-3	Type = 3
4-6	Reserved
7	PF pad field = 0 indicating no padding follows

SRCNODE (2) : B The identification of the source node's Routing Layer, containing the value ID as set by Network Management in the SELF parameters.

RES (1) : B A reserved field of 1 byte

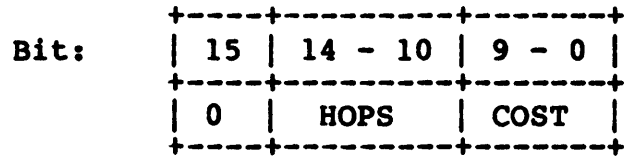
SEGMENT is of the FORM:



COUNT (2) : B The number of IDs in the RTGINFO segment

STARTID (2) : B The first ID reported in the Routing Message, with the top 6 bits (the area bits) set to 0

RTGINFO : BM The hops and cost to a destination, in the format:



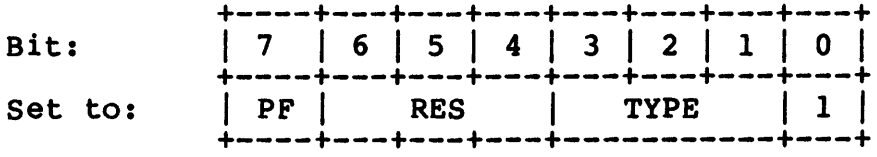
CHECKSUM (2) : B A check on the routing database, and a check to ensure Phase IV Routing Messages not be mistaken for Phase III Routing Messages, as well as a check of the message. It is a one's complement add starting with the first SEGMENT and continuing until the CHECKSUM.

To ensure that a Phase IV Routing Message will be distinguished from a Phase III Routing Message, the sum on the Phase IV Routing Message is initialized to 1.

9 LEVEL 2 ROUTING MESSAGE

A Level 2 Routing Message contains one or more segments, each segment referring to COUNT areas starting with STARTAREA.

FLAGS (1) : BM The Routing Layer control flag, with the following format:



Bit	Definition
1-3	Type = 4
4-6	Reserved
7	PF pad field = 0 indicating no padding follows

SRCNODE (2) : B The identification of the source node's Routing Layer, containing the value ID as set by Network Management in the SELF parameters.

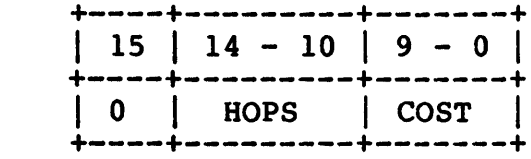
RES (1) : B A reserved field of 1 byte

SEGMENT is of the FORM:

COUNT (2) : B The number of areas in the RTGINFO segment

STARTAREA (2) : B The first area reported in the Routing Message, with the top 10 bits 0 (area is a 6 bit quantity)

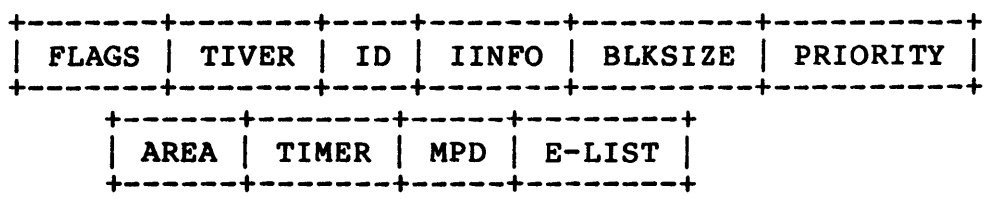
RTGINFO : BM Hops and cost to a destination area, in the format:



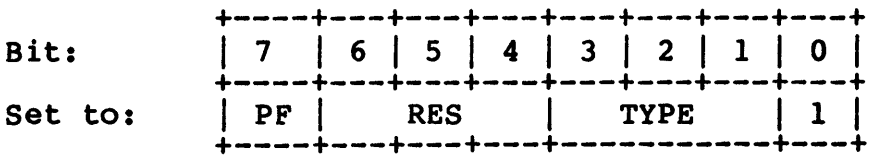
CHECKSUM (2) : B A check on the routing database, and a check to ensure Phase IV Routing Messages not be mistaken for Phase III Routing Messages, as well as a check of the message. It is a one's complement add starting with the first SEGMENT and continuing until the CHECKSUM.

To ensure that a Phase IV Routing Message will be distinguished from a Phase III Routing Message, the sum on the Phase IV Routing Message is initialized to 1.

10 ETHERNET ROUTER HELLO MESSAGE



FLAGS (1) : BM The Routing Layer control flag, with the following format:



Bit	Definition
0	1 indicates Control Packet
1-3	Type = 5
4-6	Reserved
7	PF pad field = 0 indicating no padding follows

TIVER (3) : B — The Routing Layer version, with the following format:
Byte 1 -- version number (2)
Byte 2 -- ECO number (0)
Byte 3 -- user ECO number (0)

ID (6) : B — The system ID of the transmitter

IINFO (1) : BM — flags

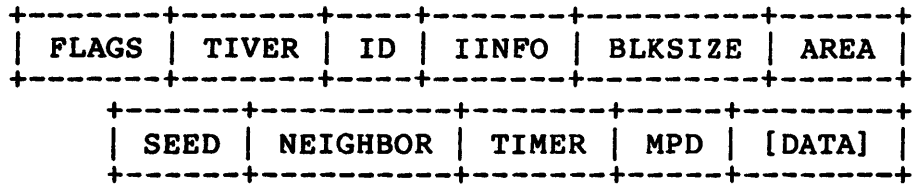
Bit	Definition
0,1	2=level 1 router 1=level 2 router
2	Verification Required flag (0 on the Ethernet)
3	Reject Flag, Reserved
4	Verification Failed, Reserved
5	No Multicast Traffic Accepted 0=accepts multicast
6	Blocking Requested Flag (0 on Ethernet)
7	Reserved

BLKSIZE (2) : B	Maximum Data Link Layer receive block size -- (Includes Routing header, excludes Data Link header)
PRIORITY (1) : B	Router's priority
AREA (1) : B	Reserved
TIMER (2) : B	Hello Timer in seconds
MPD (1) : B	Reserved
E-LIST (I-244)	List of router states for logical Ethernets on this physical Ethernet The format of each list item is:
NAME (7) : B	Logical Ethernet name, reserved
R/S-LIST (I-236)	List of router/state pairs. The format of each list item is:
ROUTER (6) : B	Router ID
PRISTATE (1) : BM	Priority and state
Bit 7: State :	1 means known 2-way, 0 otherwise
Bits 0-6: Priority:	This router's priority

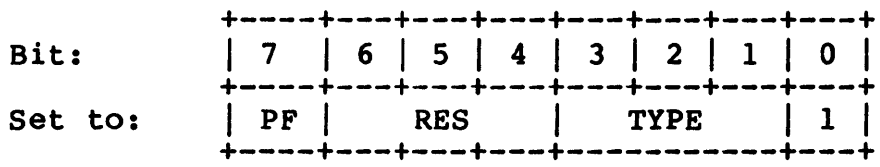
Note: E-LIST will always contain a single entry of the format:

1. NAME = 0
2. R/S-LIST = list of router/state pairs

11 ETHERNET ENDNODE HELLO MESSAGE



FLAGS (1) : BM The Routing Layer control flag, with the following format:



Bit	Definition
0	1 indicates Control Packet
1-3	Type = 6
4-6	Reserved
7	PF pad field = 0 indicating no padding follows

TIVER (3) : B The version, with the following format:

Byte 1 -- version number (2)
Byte 2 -- ECO number (0)
Byte 3 -- user ECO number (0)

ID (6) : B ID of the transmitting node. The first 4 bytes must be set to HIORD. The bottom 2 bytes are the Phase IV address (ID) assigned to the node.

IINFO (1) : BM — flags

Bit	Definition
0,1	The node type (value 3, for endnode)
2	Verification Required flag (0 on the Ethernet)
3	Reject Flag, Reserved
4	Verification Failed, Reserved
5	No Multicast Traffic Accepted 0=accepts multicast
6	Blocking Requested Flag (0 on Ethernet)
7	Reserved

BLKSIZE (2) : B — Maximum Data Link Layer receive block size -- (Includes Routing header, excludes Data Link header)

AREA (1) : B — Reserved

SEED (8) : B — The verification seed (0)

NEIGHBOR (6) : B — Neighbor's system ID, ID of Designated Router on Ethernets (0 if no Designated Router)

TIMER (2) : B — Hello Timer, in seconds

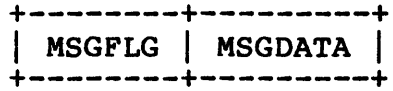
MPD (1) : B — Reserved

DATA (I-128) : B — A sequence of 0 to 128 bytes of data used to test the circuit. Each byte is 252 octal.

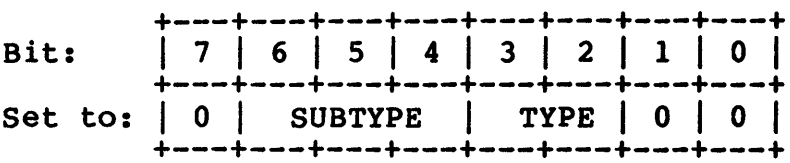
NSP (ECL) MESSAGE FORMATS

1 GENERAL MESSAGE FORMAT

In general, NSP messages have the following format:



MSGFLG (EX) : BM Is a group of fields describing the characteristics of the message. The MSGFLG format is:



TYPE (2B) : B Is the message type (binary)

0 data message
1 acknowledgment message
2 control message
3 reserved

SUBTYPE (3B) : B Is the message subtype - used to modify TYPE field.

Type	Subtype (Bits)	Bit set to/ Meaning
0	4	0 Data Segment
		1 Interrupt or Link Service
	5	1 Beginning-of-Message segment (bit 4 = 0)
	6	1 End-of-Message segment (bit 4 = 0)
	5	0 Link Service (bit 4 = 1)
	5	1 Interrupt (bit 4 = 1)
	6	0 Reserved (bit 4 = 1)
1	4-5	0 Data Acknowledgment
		1 Other-Data Acknowledgment
		2 Connect Acknowledgment
		3 Reserved
	6	0 Reserved
2	4-6	control type (binary):
		0 No Operation (included for compatibility with NSP 3.1)
		1 Connect Initiate
		2 Connect Confirm
		3 Disconnect Initiate
		4 Disconnect Confirm
		5 Reserved-Phase II node init
		6 Retransmitted Connect Init.
		7 Reserved

MSGDATA Is the remainder of an NSP message

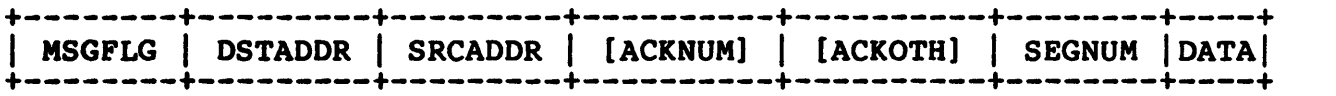
2 DATA MESSAGES

There are three types of data messages:

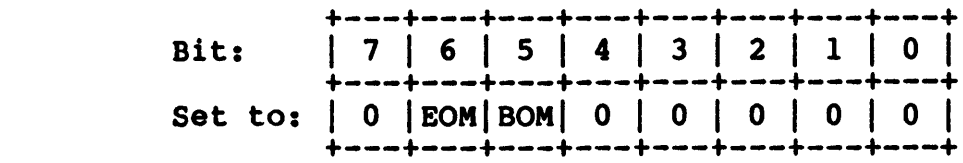
1. Data Segment Messages
2. Interrupt Messages
3. Link Service Messages

2.1 Data Segment Message

A Data Segment message has the following format:



MSGFLG (EX) : BM Represents the message identifier. The format is:



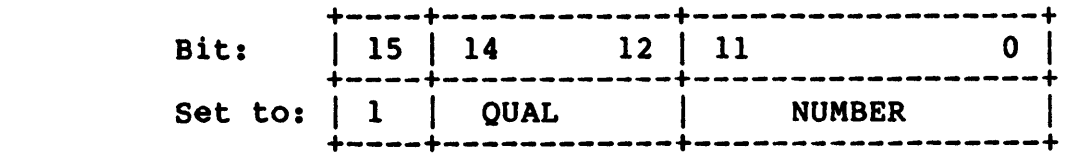
EOM (1B) : BM Is the end-of-message indicator
- 0 not-end-of-message
- 1 end-of-message

BOM (1B) : BM Is the beginning-of-message indicator
- 0 not-beginning-of-message
- 1 beginning-of-message

DSTADDR (2) : B Is the logical link destination address.

SRCADDR (2) : B Is the logical link source address.

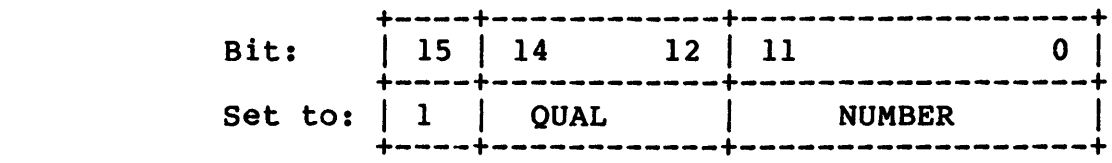
ACKNUM (2) : BM Is the number of the last NSP Data Segment message successfully received and a positive acknowledgment (ACK) or a negative acknowledgment (NAK). This field is optional. Its presence is indicated by bit 15 being set. The format for this field is as follows:



QUAL (3B) : B Is an acknowledgment qualifier.
- 0 ACK
- 1 NAK
- 2-7 reserved

NUMBER (12B) : B Is the number of the message being acknowledged.

ACKOTH (2) : BM — Is the number of the last NSP Other Data message successfully received and a positive acknowledgment (ACK) or a negative acknowledgment (NAK). This field is optional. Its presence is indicated by bit 15 being set. The format for this field is as follows:

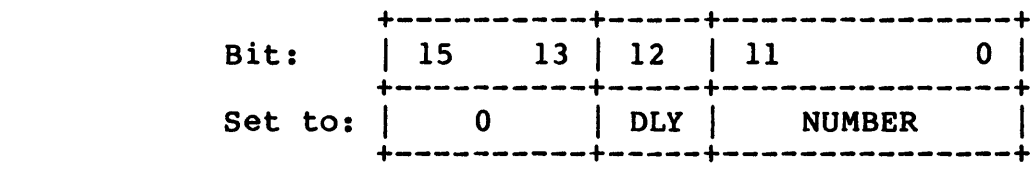


QUAL (3B) : B — Is an acknowledgment qualifier.

0,1 reserved
2 cross sub-channel ACK
3 cross sub-channel NAK
4-7 reserved

NUMBER (12B) : B — Is the number of the message being acknowledged.

SEGNUM (2) : BM — Is the number of this Data Segment message. The format for this field is:



DLY (1B) : BM is the ACK delay flag
0 - do not delay ACK
1 - delayed ACK allowed

DATA — Is the data to be sent over a logical link. This field is transparent and may use all 8 bits of each byte. The length of the data field is ascertained from the total length of the Data Segment message and consists of all bytes in the message after the SEGNUM field.

2.2 Interrupt Message

The Interrupt message has the following form:

MSGFLG	DSTADDR	SRCADDR	[ACKNUM]	[ACKDAT]	SEGNUM	DATA

MSGFLG (EX) : BM Is the message identifier. The format of this field is:

Bit:	7	6	5	4	3	2	1	0
Set to:	0	0	1	1	0	0	0	0

DSTADDR (2) : B Is the logical link destination address.

SRCADDR (2) : B Is the logical link source address.

ACKNUM (2) : BM Is the number of the last NSP Interrupt or Link Service message successfully received and a positive acknowledgment (ACK) or a negative acknowledgment (NAK). This field is optional. Its presence is indicated by bit 15 being set. The format for this field is as follows:

Bit:	15	14 12	11 0
Set to:	1	QUAL	NUMBER

QUAL (3B) : B Is an acknowledgment qualifier.

0 ACK
1 NAK
2-7 reserved

NUMBER (12B) : B Is the number of the message being acknowledged.

ACKDAT (2) : BM — Is the number of the last NSP Data Segment message successfully received and a positive acknowledgment (ACK) or a negative acknowledgment (NAK). This field is optional. Its presence is indicated by bit 15 being set. The format for this field is as follows:

Bit:	15	14 12	11 0
Set to:	1	QUAL	NUMBER

QUAL (3B) : B — Is an acknowledgment qualifier.

0,1 reserved
2 cross sub-channel ACK
3 cross sub-channel NAK
4-7 reserved

NUMBER (12B) : B — Is the number of the message being acknowledged.

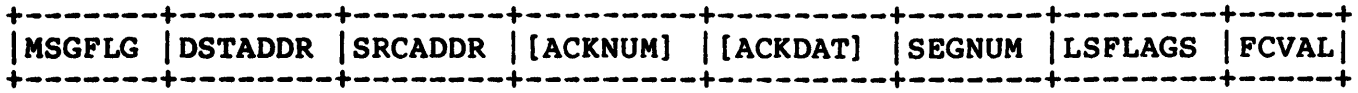
SEGNUM (2) : BM — Is the number of this Interrupt message. The format for this field is:

Bit:	15 12	11 0
Set to:	0	NUMBER

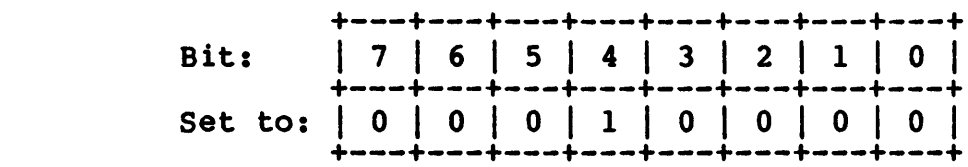
DATA — Is the interrupt data. This field is transparent and may use all 8 bits of each byte. The length of the data field is ascertained from the total length of the Interrupt message and consists of all bytes in the message after the SEGNUM field. Interrupt data may be no longer than 16 bytes.

2.3 Link Service Message

The Link Service message has the following form:



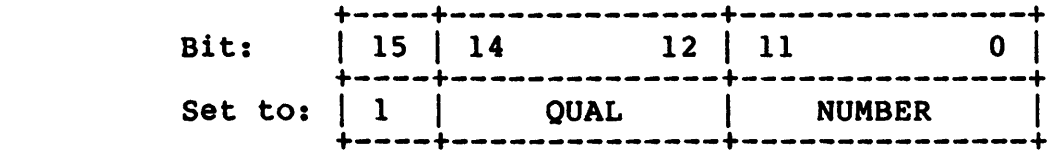
MSGFLG (EX) : BM — Is the message identifier. The format of this field is:



DSTADDR (2) : B — Is the logical link destination address.

SRCADDR (2) : B — Is the logical link source address.

ACKNUM (2) : BM — Is the number of the last NSP Interrupt or Link Service message successfully received and a positive acknowledgment (ACK) or a negative acknowledgment (NAK). This field is optional. Its presence is indicated by bit 15 being set. The format for this field is as follows:

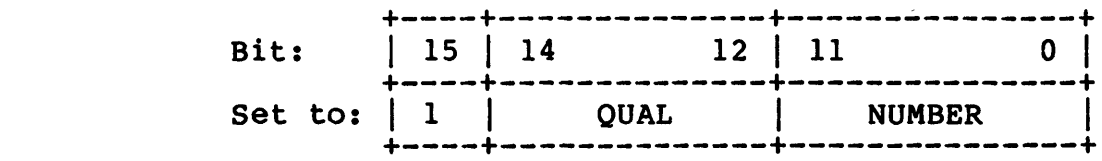


QUAL (3B) : B — Is an acknowledgment qualifier.

0 ACK
1 NAK
2-7 reserved

NUMBER (12B) : B — Is the number of the message being acknowledged.

ACKDAT (2) : BM Is the number of the last NSP Data Segment message successfully received and a positive acknowledgment (ACK) or a negative acknowledgment (NAK). This field is optional. Its presence is indicated by bit 15 being set. The format for this field is as follows:

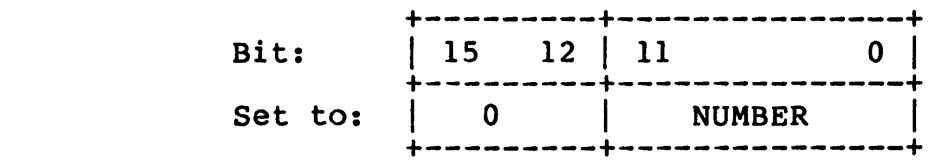


QUAL (3B) : B Is an acknowledgment qualifier.

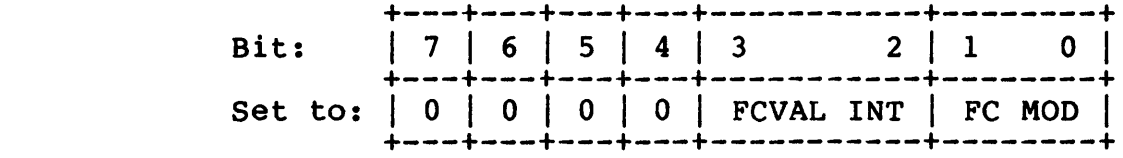
0,1 reserved
2 cross sub-channel ACK
3 cross sub-channel NAK
4-7 reserved

NUMBER (12B) : B Is the number of the message being acknowledged.

SEGNUM (2) : BM Is the number of this Link Service message. The format for this field is:



LSFLAGS (EX) : BM Is the Link Service flags. The format for this field is as follows:



FCVAL INT (2B) : B — Is the interpretation of FCVAL field

0 data segment or message request count
1 interrupt request count
2-3 reserved

FC MOD (2B) : B — Is the flow control modification. If FCVAL INT = 0, then this field has the following contents.

0 no change
1 do not send data
2 send data
3 reserved

If FCVAL INT = 1, then this field is 0 on transmit and ignored on receive.

FCVAL (1) : B — Is the number of Session Control messages, Data Segment messages, or Interrupt messages that the sender of the message can receive in addition to those previously requested by a Link Services message. This number is added to the request count which is maintained by NSP, to determine how many Session Control messages, Data Segment messages, or Interrupt messages will be transmitted via a logical link.

NOTES

1. If FCVAL INT = 0, the message is a Data Request message.

2. 2. If FCVAL INT = 1, the message is an Interrupt Request message.

3. The transmit request count for segment flow control may be negative. (Negative values are presented in 2's complement form in the FCVAL field.)

4. If FCVAL is for Session Control message or Interrupt message flow control, the count must be positive. Use 0 if there is to be no change in the count.

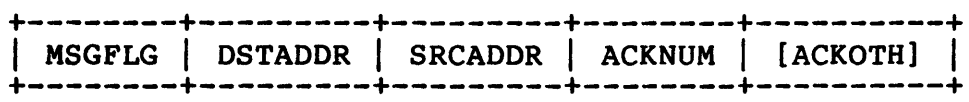
3 ACKNOWLEDGMENT TYPES

There are three types of acknowledgment messages:

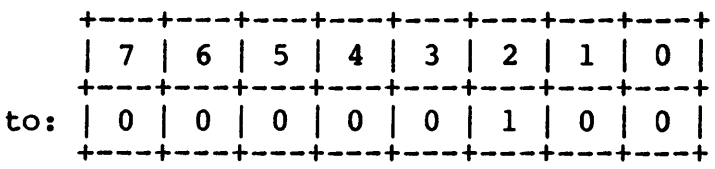
1. Data Acknowledgment Message
2. Other-Data Acknowledgment Messages
3. Connect Acknowledgment Messages

3.1 Data Acknowledgment Message

The Data Acknowledgment message has the following form:



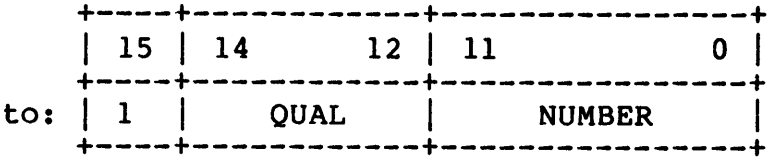
MSGFLG (EX) : BM Is the message identifier. The format of this field is:



DSTADDR (2) : B Is the logical link destination address.

SRCADDR (2) : B Is the logical link source address.

ACKNUM (2) : BM Is the number of the last NSP Data Segment message successfully received and a positive acknowledgment (ACK) or a negative acknowledgment (NAK). This field is required. The format for this field is as follows:



QUAL (3B) : B Is an acknowledgment qualifier.

0 ACK
1 NAK
2-7 reserved

NUMBER (12B) : B Is the number of the message being acknowledged.

ACKOTH (2) : BM Is the number of the last NSP Other Data message successfully received and a positive acknowledgment (ACK) or a negative acknowledgment (NAK). This field is optional. Its presence is indicated by bit 15 being set. The format for this field is as follows:

Bit:	15	14 12	11 0
Set to:	1	QUAL	NUMBER

QUAL (3B) : B Is an acknowledgment qualifier.

0,1 reserved
2 cross sub-channel ACK
3 cross sub-channel NAK
4-7 reserved

NUMBER (12B) : B Is the number of the message being acknowledged.

3.2 Other-Data Acknowledgment Message

The Other-Data Acknowledgment message acknowledges Interrupt and Link Service messages.

It has the following form:

MSGFLG	DSTADDR	SRCADDR	ACKNUM	[ACKDAT]

MSGFLG (EX) : BM — Is the message identifier. The format of this field is:

Bit:	7	6	5	4	3	2	1	0
Set to:	0	0	0	1	0	1	0	0

DSTADDR (2) : B — Is the logical link destination address.

SRCADDR (2) : B — Is the logical link source address.

ACKNUM (2) : BM — Is the number of the last NSP Interrupt or Link Service message successfully received and a positive acknowledgment (ACK) or a negative acknowledgment (NAK). This field is required. The format for this field is as follows:

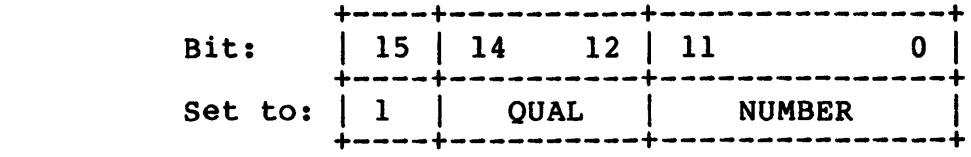
Bit:	15	14 12	11 0
Set to:	1	QUAL	NUMBER

QUAL (3B) : B — Is an acknowledgment qualifier.

0 ACK
1 NAK
2-7 reserved

NUMBER (12B) : B — Is the number of the message being acknowledged.

ACKDAT (2) : BM Is the number of the last NSP Data Segment message successfully received and a positive acknowledgment (ACK) or a negative acknowledgment (NAK). This field is optional. Its presence is indicated by bit 15 being set. The format for this field is as follows:



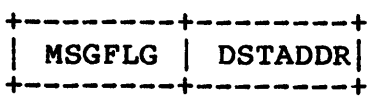
QUAL (3B) : B Is an acknowledgment qualifier.

0,1 reserved
2 cross sub-channel ACK
3 cross sub-channel NAK
4-7 reserved

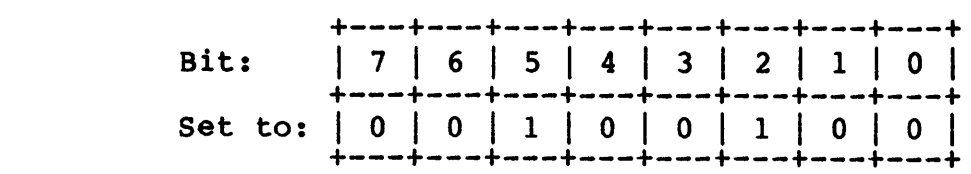
NUMBER (12B) : B Is the number of the message being acknowledged.

3.3 Connect Acknowledgment Message

The Connect Acknowledgment message has the following form:



MSGFLG (EX) : BM Is the message identifier. The format of this field is:



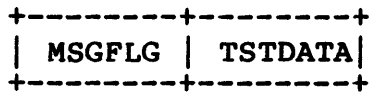
DSTADDR (2) : B Is the logical link destination address.

4 CONTROL MESSAGES

There are five types of control messages:

1. No Operation Messages
2. Connect Initiate Messages
3. Connect Confirm Message
4. Disconnect Initiate Messages
5. Disconnect Confirm Messages

4.1 No Operation Message



MSGFLG (EX) : BM Is the message identifier. The format of this field is:

Bit:	7	6	5	4	3	2	1	0
Set to:	0	0	0	0	1	0	0	0

TSTDATA Is any data.

4.2 Connect Initiate and Retransmitted Connect Initiate Messages

The Connect Initiate and the Retransmitted Connect Initiate messages have the following form:

MSGFLG	DSTADDR	SRCADDR	SERVICES	INFO	SEGSIZE	DATA-CTL

MSGFLG (EX) : BM Is the Connect Initiate message identifier. The format of this field is:

Bit:	7	6	5	4	3	2	1	0
Set to:	0	0	0	1	1	0	0	0

or

MSGFLG (EX) : BM Is the Retransmitted Connect Initiate message identifier. The format of this field is:

Bit:	7	6	5	4	3	2	1	0
Set to:	0	1	1	0	1	0	0	0

DSTADDR (2) : B Is the destination logical link address. This address will be 0 to allow the receiving NSP to assign a number dynamically.

SRCADDR (2) : B Is the source logical link address. This number is assigned by the sending NSP and will be used by the destination to address all messages for this logical link. The value 0 is illegal.

SERVICES (EX) : BM The requested services. The format for this field is as follows:

Bit:	7	6	5	4	3	2	1	0
Set to:	0	0	0	0	FCOPT		0	1

FCOPT (2B) : B Are the flow control options.

0 none
1 segment request count
2 Session Control message request count
3 reserved

INFO (EX) : BM Is the information. The format for this field is as follows:

Bit:	7	6	5	4	3	2	1	0
Set to:	0	0	0	0	0	0	VER	

VER (2B) : B Is the NSP version.

0 version 3.2
1 version 3.1
2 version 4.0
3 reserved

SEGSIZE (2) : B Is the maximum size (in bytes) of the data in a Data Segment that can be received on this logical link.

DATA-CTL Is the Connect Initiate data field. The length of this field is ascertained from the total length of the Connect Initiate message and consists of all bytes in the message after the SEGSIZE field.

4.3 Connect Confirm Message

The Connect Confirm message has the following form:

MSGFLG	DSTADDR	SRCADDR	SERVICES	INFO	SEGSIZE	DATA-CTL

MSGFLG (EX) : BM Is the message identifier. The format of this field is:

Bit:	7	6	5	4	3	2	1	0
Set to:	0	0	1	0	1	0	0	0

DSTADDR (2) : B Is the destination logical link address. This will not be 0. It is the value of the SRCADDR field from the Connect Initiate message.

SRCADDR (2) : B Is the source logical link address. This number is assigned by the sending NSP and will be used to address all messages for this logical link. The value 0 is illegal.

SERVICES (EX) : BM Are the requested services. The format for this field is as follows:

Bit:	7	6	5	4	3	2	1	0
Set to:	0	0	0	0	FCOPT		0	1

FCOPT (2B) : B Are the flow control options.

0 none
1 segment request count
2 Session Control message request count
3 reserved

INFO (EX) : BM Is the information. The format for this field is as follows:

Bit:	7	6	5	4	3	2	1	0
Set to:	0	0	0	0	0	0	VER	

VER (2B) : B Is the NSP version.

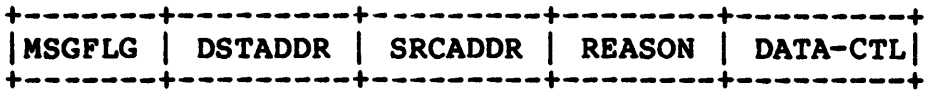
0 version 3.2
1 version 3.1
2 version 4.0
3 reserved

SEGSIZE (2) : B Is the maximum size (in bytes) of the data in a Data Segment that can be received on this logical link.

DATA-CTL (I-16) : B Is user-supplied data.

4.4 Disconnect Initiate Message

The Disconnect Initiate message has the following form:



MSGFLG (EX) : BM — Is the message identifier. The format of this field is:

Bit:	7	6	5	4	3	2	1	0
Set to:	0	0	1	1	1	0	0	0

DSTADDR (2) : B — Is the logical link destination address.

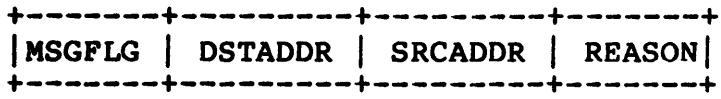
SRCADDR (2) : B — Is the logical link source address.

REASON (2) : B — Is the first two bytes of Session Control disconnect data.

DATA-CTL (I-16) : B — Is the remaining bytes of Session Control disconnect data.

4.5 Disconnect Confirm Message

A Disconnect Confirm message has the following form:



MSGFLG (EX) : BM Is the message identifier. The format of this field is:

Bit:	7	6	5	4	3	2	1	0
Set to:	0	1	0	0	1	0	0	0

DSTADDR (2) : B Is the logical link destination address.

SRCADDR (2) : B Is the logical link source address.

REASON (2) : B Is the disconnect reason.

NOTES

1. **If REASON = 1, the message is a No Resources message.**
2. **If REASON = 42, the message is a Disconnect Complete message.**
3. **If REASON = 41, the message is a No Link Terminate message.**

SESSION CONTROL LAYER MESSAGES

INTRODUCTION

Session Control's message protocol defines messages sent on a logical link as connect data, reject data, and disconnect data.

The notation used is the same as that used in other DIGITAL Network Architecture specifications.

1 CONNECT DATA

A Connect Data message has the following form:

DSTNAME SRCNAME MENUVER RQSTRID PASSWRD ACCOUNT USRDATA

DSTNAME (*-19) : Is the destination end-user name (see below)

SRCNAME (*-19) : Is the source end-user name (see below)

MENUVER (EX) : BM Is the field format and version information:

Bit 0=1 RQSTRID, PASSWRD, ACCOUNT fields included

Bit 1=1 USRDATA field included

Bits 2-4=0 Reserved

Bits 5,6 Is the Session Control version:

0 = Version 1.0
1-3 = Reserved

RQSTRID (I-39) : B Is the source user identification for access verification

PASSWRD (I-39) : B Is the access verification password

ACCOUNT (I-39) : B Is the link or service account data

USRDATA (I-16) : B Is the end-user connect data

An end user name has the following form:

FORMAT OBJTYPE NAME

FORMAT (1) : B Is the end-user name format:

0 = Format 0
1 = Format 1
2 = Format 2
3-255 = Reserved

OBJTYPE (1) : B Is the object type. It may not be 0 for format 0; it must be 0 for formats 1 and 2.

For format 0:

NAME (null) Is not present

For format 1:

NAME (I-16) : B Is an end-user descriptor

For format 2:

NAME Is GRPCODE, USRCODE, DESCRPT

GRPCODE (2) : B Is a group code

USRCODE (2) : B Is a user code

DESCRPT (I-12) : B Is an end-user descriptor

2 REJECT AND DISCONNECT DATA

A Reject or Disconnect message has the following form:

REASON DATA-CTL

REASON (2) : B A reason code (see Section 6.2)

DATA-CTL (I-16) : B User data. The length of this field is ascertained from the total length of reject or disconnect data received from Network Services. It may be null.

NETWORK INFORMATION AND CONTROL EXCHANGE PROTOCOL

1 NETWORK MANAGEMENT MESSAGES

This section describes the NICE and Event Logging Messages, the NICE response message format, the NICE connect and accept data format, and the Logical Link Message format.

NICE is a command-response protocol. Because the Network Management layer is built on top of the End Communication and Data Link layers, which provide logical links that guarantee sequential and error-free data delivery, NICE does not have to handle error recovery.

In the message descriptions that follow, any unused bits or bytes are to be reserved and set to zero to allow compatibility with future implementations. Conditions such as non-zero reserved areas and unrecognized codes or unused bytes at the end of a field or message should be treated as errors, and no operation should be performed other than an appropriate error response.

The entire message should be parsed and checked for validity before any operation is performed.

The method for indicating that a function should be executed on all parameters in the data base for a particular entity (NCP ALL option) is to not include any parameters in the NICE function request message.

Parameters in command and response messages must be in ascending order by parameter type number, except that qualifiers must precede the parameters they qualify. A parameter of the same type may be repeated for parameters that compose a list.

Qualifiers are restricted to one of the same parameter types per command or response message. If qualified parameters appear in a message the qualifier must appear in the same message. Unqualified parameters may or may not be included in a message with a qualifier. All qualified parameters in a message with a qualifier must be associated with that qualifier. In a sequence of multiple return messages, when qualifiers are nested, the outer level qualifier is not repeated until it changes.

2 NICE FUNCTION CODES

The NICE protocol performs the following message functions. The last one is for system-specific commands not specified in this document.

Function Code	NICE Function
15	Request downline load
16	Request upline dump
17	Trigger bootstrap
18	Test
19	Change parameter
20	Read information
21	Zero counters
22	System-specific function

3 REQUEST DOWNLINE LOAD MESSAGE FORMAT

FUNCTION CODE | OPTION | NODE | CIRCUIT | PARAMETER ENTRIES

FUNCTION CODE (1) : B = 15

OPTION (1) BM Is one of the following options:

Option bit	Value/Meaning
0-2	0 = Identify target by node-id. 3 = Identify target by circuit-id.

NODE — Is the target node identification as key into default's database (present only if option = 0). Plural node's options are not allowed.

CIRCUIT — Is the circuit identification. Plural circuits options not allowed. Present only if option = 3.

PARAMETER ENTRIES — are zero or more of PARAMETER ENTRY consisting of:

DATA ID | DATA

where:

DATA ID (2) : B — Is the parameter type number

DATA — Is the parameter data

NOTE

The following node parameters are allowed:

ADDRESS
CPU
DIAGNOSTIC FILE
HOST
LOAD FILE
NAME
PHYSICAL ADDRESS
SECONDARY LOADER
SERVICE DEVICE
SERVICE PASSWORD
SERVICE CIRCUIT (allowed only if option = 0)
SOFTWARE IDENTIFICATION
SOFTWARE TYPE
TERTIARY LOADER

4 REQUEST UPLINE DUMP MESSAGE FORMAT

FUNCTION CODE | OPTION | NODE | CIRCUIT | PARAMETER ENTRIES

FUNCTION CODE (1): B = 16

OPTION (1) : BM — Is one of the following options:

Option bits	Value/Meaning
0-2	0 = Identify target by node-id. 3 = Identify target by circuit-id.

NODE — Identifies the node to be dumped (present only if option = 0).

CIRCUIT — Specifies the circuit over which to dump (present only if option = 3).

PARAMETER ENTRIES — Are zero or more of PARAMETER ENTRY consisting of:

DATA ID | DATA

where:

DATA ID (2) : B — Is the parameter type number

DATA — Is the parameter data

NOTE

The parameters are selected from the node parameters. Only certain parameters are allowed in the dump message. They are:

- **DUMP ADDRESS**
- **DUMP COUNT**
- **DUMP FILE**
- **PHYSICAL ADDRESS**
- **SECONDARY DUMPER**
- **SERVICE CIRCUIT (allowed only if option = 0)**
- **SERVICE PASSWORD**

5 TRIGGER BOOTSTRAP MESSAGE FORMAT

```
FUNCTION   OPTION   NODE   CIRCUIT    PARAMETER
  CODE                                  ENTRIES
```

where:

FUNCTION CODE (1): B = 17

OPTION (1) : BM Is one of the following options:

Option bits	Value/Meaning
0-2	0 = Identify target by node-id. 3 = Identify target by circuit-id.

NODE — Identifies the node to trigger boot on (present only if option = 0).

CIRCUIT — Identifies the circuit over which to trigger the boot (present only if option = 3).

PARAMETER ENTRIES — Are zero or more of PARAMETER ENTRY consisting of:

```
DATA     DATA
 ID
```

where:

DATA ID (2) : B Is the parameter type number

DATA Is the parameter data

NOTE

The parameters are selected from the node parameters. Only certain parameters are allowed in the trigger message. They are:

PHYSICAL ADDRESS
SERVICE CIRCUIT (allowed only if option = 0)
SERVICE PASSWORD

6 TEST MESSAGE FORMAT

FUNCTION CODE | OPTION | NODE | USER | PASSWORD | ACCOUNTING | LINK | PARAMETER ENTRIES

where:

FUNCTION CODE (1): B = 18

OPTION (1) : BM Is one of the following options:

Option bits	Value/Meaning
0-2	0 = Node type loop test 1 = Line loop test 3 = Circuit loop test

If node type loop test:

7	0 = Default access control 1 = Access control included

For node type loop tests only (option 0), four parameters are as follows:

NODE — Identifies the node to loopback the test block in node-id format. Plural node options are not allowed.

USER (I-39): A — Is the user-id to use when connecting to node. Present only if option bit 7 = 1.

PASSWORD (I-39): A — Is the password to use when connecting to node. Present only if option bit 7 = 1.

ACCOUNTING (I-39):A — Is the accounting information to use when connecting to node. Present only if option bit 7 = 1.

For line or circuit tests only (option 1 or 3), one parameter is as follows:

LINK	Identifies the link to send the test on in circuit- or line-id format. Plural options not allowed.
PARAMETER ENTRIES	Are zero or more of PARAMETER ENTRY, consisting of:

DATA ID DATA

where:

DATA ID (2) : B Is the parameter type number

DATA Is the parameter data

NOTE

The parameters are selected from the node parameters. Only certain parameters are allowed in the test message. They are:

LOOP ASSISTANT NODE
LOOP ASSISTANT PHYSICAL ADDRESS
LOOP COUNT
LOOP HELP
LOOP LENGTH
LOOP NODE
LOOP WITH
PHYSICAL ADDRESS

7 CHANGE PARAMETER MESSAGE FORMAT

FUNCTION CODE	OPTION	ENTITY ID	PARAMETER ENTRIES

where:

FUNCTION CODE (1): B = 19

OPTION (1): BM — Is one of the following options:

Bits	Meaning
7	0 = Change volatile parameters.
	1 = Change permanent parameters.
6	0 = Set/define parameters.
	1 = Clear/purge parameters.
0-2	Entity type

ENTITY ID — Identifies the particular entity

PARAMETER ENTRIES — Are zero or more of PARAMETER ENTRY consisting of:

DATA ID	DATA

where:

DATA ID (2) : B — Is the parameter type number

DATA — Is the parameter data

NOTE

The DATA field is not present when option bit 6 is set unless the parameter is a qualifier rather than a parameter that is to be cleared.

8 READ INFORMATION MESSAGE FORMAT

FUNCTION CODE	OPTION	ENTITY ID	PARAMETER ENTRIES

FUNCTION CODE (1): B = 20

OPTION (1): BM	Bits	Meaning
	7	0 = Read volatile parameter 1 = Read permanent parameter
	4-6	Information type as follows: 0 = Summary 1 = Status 2 = Characteristics 3 = Counters 4 = Events
	0-2	Entity type

ENTITY ID	Identifies the particular entity
PARAMETER ENTRIES	0 or more parameter entries, formatted as for change parameter message. These are limited to:

CIRCUIT ADJACENT NODE
LOGGING SINK NODE
MODULE X25-ACCESS NETWORK
MODULE X25-PROTOCOL DTE GROUP
MODULE X25-SERVER DESTINATION
NODE CIRCUIT

9 ZERO COUNTERS MESSAGE FORMAT

```
FUNCTION     OPTION     ENTITY     PARAMETER
  CODE                    ID        ENTRIES
FUNCTION CODE (1):  B =  21

OPTION (1):  BM         Bits          Meaning

                         7        1 = Read and zero
                                  0 = Zero only
                        0-2       Entity type (Circuit, line,
                                  module, or node only).

ENTITY ID               Identifies the particular  entity,  if  required

PARAMETER ENTRIES       0 or more parameter entries,  formatted  as  for
                        change  parameter message.  These are limited to:
                                     MODULE X25-PROTOCOL DTE
```

10 NICE SYSTEM-SPECIFIC MESSAGE FORMAT

```
FUNCTION      SYSTEM      REMAINDER
  CODE         TYPE
```

where:

FUNCTION CODE (1) : B = 22

SYSTEM TYPE (1) : B Represents the type of operating system command to which command is specific.

Value	System
Value	System
1	RSTS
2	RSX family
3	TOPS-10/20
4	VMS
5	RT
6	CT
7	Communications Server

REMAINDER (*) : B Consists of data, depending on system-specific requirements.

11 NICE RESPONSE MESSAGE FORMAT

RETURN CODE	ERROR DETAIL	ERROR MESSAGE	ENTITY ID	TEST DATA	DATA BLOCK

RETURN CODE (1) : B Is one of the standard NICE return codes

[ERROR DETAIL] (2) : B Is more detailed error information according to the error code (e.g., a parameter type). Zero if not applicable. If applicable but not available, its value is 65,535 (all bits set). In this case it is not printed. Applicable only if 0 > RETURN CODE > -128.

[ERROR MESSAGE] (I-255) : A Is a system-dependent error message that may be output in addition to the standard error message.

[ENTITY ID] Identifies a particular entity if operation is on plural entities, or operation is read information or read and zero counters. If the entity is the executor node, bit 7 of the name length is set.

[TEST DATA] (2) : B Is the information resulting from a test operation (Test message only). This is only required if a test failed and if data is relevant, or a data block is present in this message. It is UNLOOPED COUNT or MAXIMUM LOOP DATA, depending on ERROR DETAIL.

[DATA BLOCK] Is one of the data blocks returned for read information message or read and zero message. For Test messages it may contain the parameter PHYSICAL ADDRESS.

If a response message is short terminated after any field, the existing fields may still be interpreted according to standard format. This means, for example, that a single byte return is to be interpreted as a return code.

Responses to messages not noted as exceptions above are single responses indicating return code, error detail, and error message.

A success response to a request for plural entities is indicated by a return code of only 2 with no other fields, followed by a separate response message for each entity. Each of these messages contains the basic response data (return code, error detail, and error message) and the entity id. A return code of -128 indicates the end of multiple responses.

12 NICE CONNECT INITIATE AND CONNECT ACCEPT DATA FORMATS

The first three bytes of the connect initiate and connect accept data are:

VERSION	DEC	USER
	ECO	ECO

VERSION (1) : B Is the version number

DEC ECO (1) : B Is the DIGITAL ECO number

USER ECO (1) : B Is the user ECO number

13 EVENT MESSAGE BINARY DATA FORMAT

This section describes the generalized binary format of event data. It applies to messages on logical links and, as much as possible, to files. The buffer size for event messages is 200 bytes.

The format of an event logging message is:

FUNCTION CODE	SINK FLAGS	EVENT CODE	EVENT TIME	SOURCE NODE	EVENT ENTITY	EVENT DATA

FUNCTION CODE (1) : B = 1, meaning event log

SINK FLAGS (1) : BM Are flags indicating which sinks are to receive a copy of this event, one bit per sink. The bit assignments are:

Bit	Sink
0	Console
1	File
2	Monitor

EVENT CODE (2) : BM Identifies the specific event as follows:

Bits	Meaning
0-4	Event type
6-14	Event class

EVENT TIME Is the source node date and time of event processing. Consists of:
JULIAN SECOND MILLISECOND
HALF DAY

JULIAN HALF DAY (2) : B — Number of half days since 1 Jan 1977 and before 9 Nov 2021 (0-32767). For example, the morning of Jan 1, 1977 is 0.

SECOND (2) : B — Second within current half day (0-43199).

MILLISECOND (2) : B — Millisecond within current second (0-999). If not supported, high-order bit is set, remainder is clear, and field is not printed when formatted for output.

SOURCE NODE — Identifies the source node. It consists of:

NODE ADDRESS — NODE NAME

NODE ADDRESS (2) : B — Node address

NODE NAME (I-6) : A — Node name, 0 length, if none.

EVENT ENTITY — Identifies the entity involved in the event, as applicable. Consists of:

ENTITY TYPE — ENTITY ID

ENTITY TYPE (2) : B — Represents the type of entity. A -1 value indicates no entity. A value >= 0 is the entity type and is followed by the entity id in its usual format.

EVENT DATA (*) : B Is event specific data, zero or more data entries as defined for NICE data blocks, parameter types

14 LOGICAL LOOPBACK MESSAGE FORMATS

14.1 Connect Accept Data Format

MAXIMUM DATA

where:

MAXIMUM DATA (2) : B Is the maximum length, in bytes, that the Loopback Mirror can loop.

14.2 Command Message Format

FUNCTION DATA
CODE

where:

FUNCTION CODE (1) : B = 0

DATA (*) : B Is the data to loop.

14.3 Response Message Format

RETURN CODE DATA

where:

RETURN CODE (1) : B Indicates Success (1) or Failure (-1).

DATA (*) : B Is the data as received, if success.

DECnet-VAX SDL FILES

1 DLEDEF SDL FILES

```
{
{       Version 'V04-000'
{
{*****************************************************************************
{*                                                                           *
{*  COPYRIGHT (c) 1978, 1980, 1982, 1984 , 1986 BY                           *
{*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                   *
{*  ALL RIGHTS RESERVED.                                                     *
{*                                                                           *
{*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED    *
{*  ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE           *
{*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER      *
{*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY    *
{*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY       *
{*  TRANSFERRED.                                                             *
{*                                                                           *
{*  THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE     *
{*  AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT         *
{*  CORPORATION.                                                             *
{*                                                                           *
{*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS      *
{*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                  *
{*                                                                           *
{*                                                                           *
{*****************************************************************************

{++
{
{ FACILITY:  DECnet-VAX DLE driver definitions
{
{ ABSTRACT:
{
{       This file contains the DLE structure definitions.
{
{ ENVIRONMENT:
{
{       n/a
{
{--
{
{
{ AUTHOR:
{
{       Tim Halvorsen, January 1983
{
{ MODIFIED BY:
{
{       V001    TMH0001         Tim Halvorsen   21-Apr-1983
{               Add unique identifier to DWB.
{--
```

```
{+
{
{       DLE window block, which contains the context for a single DLE
{       connection.  There may be many DLE connections on a single
{       circuit, such as an Ethernet.
{
{-
MODULE $dwbdef;

AGGREGATE dwb STRUCTURE PREFIX dwb$;
        flink   ADDRESS TAG L;          { Forward link
        blink   ADDRESS TAG L;          { Backward link
        size    WORD;                   { Bytes allocated for structure
        type    BYTE;                   { Code identifying structure type
        access  BYTE;                   { IO$_ACCESS flags (see WCB definition)
        refcnt  WORD;                   { Number of accessors of window
        flags STRUCTURE TAG W;          { Flags
            run         BITFIELD MASK;  { Set if available for I/O
            dll_rbf     BITFIELD MASK;  { Set if datalink does buffered rcvs
            dll_xbf     BITFIELD MASK;  { Set if datalink does buffered xmits
            bc          BITFIELD MASK;  { Set if circuit is an Ethernet
            delete      BITFIELD MASK;  { Set if DWB is to be deallocated
            #num_flags = 5;
            fill_1 BITFIELD             { Pad out to a full word
                        LENGTH 16-#num_flags;
            END flags;
        orgucb  ADDRESS TAG L;          { Original UCB (see WCB definition)
        {
        {   The above must correspond to the WCB structure, so that normal
        {   VMS functions can operate on these windows just like WCBs.
        {   The remainder of the block from this point is DLE specific.
        {
        rcv_msg QUADWORD;               { Received message (CXB) queue
        user_rcv QUADWORD;              { User receive IRP queue
        user_xmt QUADWORD;              { User transmit IRP queue
        xmt_pnd QUADWORD;               { User transmits issued to datalink
        pid     LONGWORD;               { PID of accessor
        ucb0    LONGWORD;               { Address of ND's UCB0
        chan    WORD;                   { Channel of accessor
        path    WORD;                   { LPD of circuit
        remnod  CHARACTER LENGTH 6      { 48-bit remote node address
                        TAG G;          { used for DLE over Ethernet
        substa  BYTE;                   { DLE substate (see NMA$C_LINSS_xxx)
        irpcnt  BYTE;                   { Count of outstanding datalink I/O
        dll_ucb ADDRESS TAG L;          { Address of datalink UCB
        dll_chan WORD;                  { Datalink channel
        id      WORD;                   { Unique identifier of DWB
                                        { (used as collating sequence of list)
        CONSTANT "length" EQUALS .      { Length of structure
                        TAG C;
        END dwb;
END_MODULE $dwbdef;
```

2 NETCTL SDL FILES

```
{
{       NETCTL.SDL
{
{ Version:      'X-6'
{
{*****************************************************************************
{*                                                                           *
{*  COPYRIGHT (c) 1978, 1980, 1982, 1984, 1986 BY                            *
{*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                   *
{*  ALL RIGHTS RESERVED.                                                     *
{*                                                                           *
{*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED    *
{*  ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE           *
{*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER      *
{*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY    *
{*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY       *
{*  TRANSFERRED.                                                             *
{*                                                                           *
{*  THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE     *
{*  AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT         *
{*  CORPORATION.                                                             *
{*                                                                           *
{*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS      *
{*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                  *
{*                                                                           *
{*                                                                           *
{*****************************************************************************

{ MODIFIED BY:
{
{       X-6     PRB00X6         Paul Beck       9-DEC-1985 12:01
{               Add LOOPNODE field to CRI.
{
{       X-5     PRB00X5         Paul Beck       4-OCT-1985 14:28
{               Add CNF$V_FLG_TREE to flag a CNF as a dummy CNF used by
{               the NDI binary tree routines to represent a CNF which has
{               been deleted because it contained no information not contained
{               in the trees themselves (i.e. node name, node address)
{
{       X-4     PRB00X4         Paul Beck        9-AUG-1985 14:31
{               Add support for line and circuit NETWORK parameter
{
{       X-3     PRB00X3         Paul Beck       5-AUG-1985 16:37
{               Add support for cluster node address
{
{       V04-002 PRB0362         Paul Beck      31-Mar-1985  18:42
{               Supply Transmit Pipeline definition
{
{       V04-001 PRB0359         Paul Beck      24-Mar-1985  17:49
{               Add support for asynch line parameters
{
```

```
{       V03-018 RNG0018         Rod Gamache             10-Feb-1984
{               Add real LLI database and NDC counter block.
{               Fix up definition of a WQE.
{
{       V03-017 RNG0017         Rod Gamache             6-Feb-1984
{               Add NDI SNV (system node version) parameter.
{
{       V03-016 PRB0313         Paul Beck       6-Feb-1984  12:14
{               Fix duplicate definition of CNR$S_SEM_TAB (didn't occur with MDL)
{
{       V015    TMH0015         Tim Halvorsen           11-Jul-1983
{               Add LNI ALI (cluster node address) parameter.
{
{       V014    TMH0014         Tim Halvorsen           26-Apr-1983
{               Add PPUNA device code.
{
{       V013    RNG0013         Rod Gamache             29-Mar-1983
{               Add new CNR routine locations.
{
{       V012    TMH0012         Tim Halvorsen           14-Feb-1983
{               Add line buffer size parameter.
{
{       V011    TMH0011         Tim Halvorsen           16-Dec-1982
{               Add Ethernet protocol type to the line database.
{
{       V010    TMH0010         Tim Halvorsen           31-Aug-1982
{               Remove cell for circuit "designated router".  It is
{               now an action routine.
{
{       V009    TMH0009         Tim Halvorsen           30-Jun-1982
{               Add Phase IV line, circuit and node parameters.
{               Add UNA device type code.
{               Remove listen timer, since it's now read-only.
{               Remove NDI$C_DTY_xxx symbols, since they are now replaced
{               by ADJ$C_PTY_xxx symbols.
{
{       V008    TMH0008         Tim Halvorsen           16-Jun-1982
{               Add SPI database.
{               Add ACC_NE access code for "no external access at all".
{
{       V007    TMH0007         Tim Halvorsen           04-Apr-1982
{               Move $NDBDEF from NETUSR to here.
{               Remove spurious comments about X.25 database parameters.
{               Add "DEFAULT" action routine to CNR block.
{               Add X.25 mode byte to PLI block.
{               Remove obsolete SEM_$ constant, which is no longer
{               needed after cleanup of CNR creation macros in NETCONFIG.
{               Rename CNF$T_MASK to CNF$L_MASK.
{               Rename CNR$T_SEM_TAB to CNR$L_SEM_TAB.
{               Add KMX and X25 to DEVTRN type codes.
{
{       V03-006 ADE0033         A.Eldridge              15-Feb-1982
```

```
{               Added LNI$W_PIQ.
{
{       V03-005 ADE0032         A.Eldridge              06-Jan-1981
{               Removed the RTT parameter from the circuit database.
{
{       V03-004 ADE0031         A.Eldridge              15-Dec-1981
{               Added the CNF$V_NOEXT bit to the CNF definitions.
{
{       V03-003 ADE0030         A.Eldridge              30-Nov-1981
{               Added support for proxy logins the OBI, NDI, and LNI
{               structures.
{
```

```
{
{ Configuration Data Base Root Block  (CNR)
{
{ This block serves as the listhead for the CNFs of a particular
{ component in the configuration data base.  It contains all of the
{ component's semantics.
{
module $CNRDEF;


aggregate CNRDEF  union fill prefix CNR$;
    CNRDEF_BITS0 structure fill;
        SEM_OFF bitfield length 8;                      /* Byte offset from top of CNF to field
        SEM_TYP bitfield length 3;                      /* Field type (bit,string,byte,word etc)
        SEM_ACC bitfield length 3;                      /* Field access control
        SEM_RT bitfield;                                /* "Field" is actually an index of a routine to call
        SEM_Z bitfield;                                 /* Set if field may be zero
        SEM_MAX bitfield length 16;                     /* Max value byte or word may be assigned
    end CNRDEF_BITS0;
    CNRDEF_BITS1 structure fill;
        FILL_1 bitfield length 16 fill prefix CNRDEF tag $$;/* Advance to SEM_MAX
        SEM_SMX bitfield length 12;                     /* Field to store max size string
        SEM_TAB bitfield length 4;                      /* Holds i.d. of string parse table
    end CNRDEF_BITS1;
                                                        /*
                                                        /* Define CNF$x_SEM_TYP values
                                                        /*
    constant SEM_BIT    equals 0  prefix CNR tag $C;    /* Type = bit
    constant SEM_B      equals 1  prefix CNR tag $C;    /* Type = byte
    constant SEM_W      equals 2  prefix CNR tag $C;    /* Type = word
    constant SEM_L      equals 3  prefix CNR tag $C;    /* Type = longword
    constant SEM_STR    equals 4  prefix CNR tag $C;    /* Type = string descriptor
                                                        /*
                                                        /* Define field access control
                                                        /*
    constant ACC_RW     equals 0  prefix CNR tag $C;    /* General read/write
    constant ACC_RO     equals 1  prefix CNR tag $C;    /* Read only
    constant ACC_R      equals 1  prefix CNR tag $C;    /* Read only
    constant ACC_WO     equals 2  prefix CNR tag $C;    /* Write only (for passwords, etc)
    constant ACC_W      equals 2  prefix CNR tag $C;    /* Write only (for passwords, etc)
    constant ACC_CW     equals 3  prefix CNR tag $C;    /* Conditionally writable
    constant ACC_C      equals 3  prefix CNR tag $C;    /* Conditionally writable
    constant ACC_ER     equals 4  prefix CNR tag $C;    /* External read only (e.g., if from QIO)
    constant ACC_E      equals 4  prefix CNR tag $C;    /* External read only (e.g., if from QIO)
    constant ACC_NE     equals 5  prefix CNR tag $C;    /* No external read or write access
    constant ACC_N      equals 5  prefix CNR tag $C;    /* No external read or write access
                                                        /*
                                                        /* Define string parse table i.d.'s
                                                        /*
    constant SEM_T      equals 0  prefix CNR tag $C;    /* Transparent - all characters are legal
    constant SEM_A      equals 1  prefix CNR tag $C;    /* Upper case alpha or numerics only
    constant SEM_F      equals 2  prefix CNR tag $C;    /* Parse string as if file specification
end CNRDEF;
```

```
    aggregate CNRDEF1 structure fill prefix CNR$;
    FLINK_OVERLAY union fill;
        FLINK longword unsigned;                    /* Forward link
        COLBTE longword unsigned;                   /* Pointer to the collating tree entry for NDI CNR
    end FLINK_OVERLAY;
    BLINK_OVERLAY union fill;
        BLINK longword unsigned;                    /* Backward link
        NAMEBTE longword unsigned;                  /* Pointer to the name tree entry for NDI CNR
    end BLINK_OVERLAY;
    SIZE word unsigned;                             /* Block size
    TYPE byte unsigned;                             /* Block type (one of the NFB$C_DB_... codes)
    FLG byte unsigned;                              /* Flag bits
    SIZ_CNF word unsigned;                          /* Size of associated CNF without any string storage
    MAX_INX word unsigned;                          /* Maximum field index defined for this database
    FLD_LOCK longword unsigned;                     /* Storage for bit id of conditional write gate
    FLD_COLL longword unsigned;                     /* Storage for collating bit i.d.
    ACT_QIO longword unsigned;                      /* Ptr to QIO preprocessor for this database
    ACT_SHOW longword unsigned;                     /* Ptr to "show" QIO to a specific CNF
    ACT_DFLT longword unsigned;                     /* Ptr to "defaulting" action routine
    ACT_INSERT longword unsigned;                   /* Ptr to CNF "pre-insert" action routine
    ACT_DELETE longword unsigned;                   /* Ptr to CNF "mark-for-delete" action routine
    ACT_REMOVE longword unsigned;                   /* Ptr to CNF "post-remove" action routine
    SCANNER longword unsigned;                      /* Ptr to CNF scanner
    INSERT longword unsigned;                       /* Ptr to CNF real insertion routine
    SPCSCAN longword unsigned;                      /* Ptr to CNF special scan routine
    VEC_ACT longword unsigned dimension 16;         /* Vector of action routine pointers
    END_ACT longword unsigned;                      /* Mark the end of the vector
    VEC_MAND longword unsigned dimension 24;        /* Vector of mandatory field i.d.'s
    END_MAND longword unsigned;                     /*
    VEC_UNIQ longword unsigned dimension 16;        /* Vector of i.d.'s of fields required to be
    END_UNIQ longword unsigned;                     /* unique
    constant MAX_INX    equals 95  prefix CNR tag $C;   /* Maximum field index possible (0 indexed => 96 indexes)
    SEM_TAB_OVERLAY union fill;                     /* (Avoid duplicate definition of CNR$S_SEM_TAB)
        SEM_TAB longword unsigned;                  /* Semantic table -- 4 bytes for each of 96 indexes
        SEM_TABLE longword unsigned dimension 96;   /* ...synonym to reserve the space...
    end SEM_TAB_OVERLAY;
    constant "LENGTH" equals . prefix CNR$ tag K;   /* Structure size
    constant "LENGTH" equals . prefix CNR$ tag C;   /* Structure size
                                                    /*F        SEM_TAB,L,4*CNR$C_MAX_INX        /* Semantic table
                                                    /*-also see CNF$L_MASK definition
                                                    /*
                                                    /*  The following defines the format of each entry in the semantic t
                                                    /*

    end CNRDEF1;

end_module $CNRDEF;
```

```
module $CNFDEF;
/*
/* Configuration Data Block (CNF)
/*
/* This is a general block structure used to carry a sub-block in the
/* configuration data base of the NETACP.  The  CNF  and sub-block
/* semantics for each component type are store in the associate CNR
/* described above.
/*
aggregate CNFDEF structure fill prefix CNF$;
    FLINK_OVERLAY union fill;
        FLINK longword unsigned;                    /* Forward link
        COLBTE longword unsigned;                   /* Pointer to the collating tree entry for NDI CNF
    end FLINK_OVERLAY;
    BLINK_OVERLAY union fill;
        BLINK longword unsigned;                    /* Backward link
        NAMEBTE longword unsigned;                  /* Pointer to the name tree entry for NDI CNF
    end BLINK_OVERLAY;
    SIZE word unsigned;                             /* Block size
    TYPE byte unsigned;                             /* Block type
    FLG_OVERLAY union fill;
        FLG byte unsigned;                          /* Flags defined as follows:
        FLG_BITS structure fill;
            FLG_CNR bitfield mask;                  /* Block is actually a CNR
            FLG_DELETE bitfield mask;               /* Block is a temporary CNF or marked for delete
            FLG_ACP bitfield mask;                  /* Block is a catch-all used by the ACP
            FLG_NOEXT bitfield mask;                /* Block is used internally only. It is not to be
                                                    /* displayed to the "external" world, i.e., above the
                                                    /* $QIO interface.
            FLG_MRK1 bitfield mask;                 /* Special flags with different meanings for each
            FLG_MRK2 bitfield mask;                 /* database. These flags are defined locally in module
            FLG_MRK3 bitfield mask;                 /* NETCNFACT.MAR.
            FLG_TREE bitfield mask;                 /* Block is dummy NDI CNF owned by NETTREE
                                                    /* and created from info in the binary trees (BTE)
        end FLG_BITS;
    end FLG_OVERLAY;
    OFF_FREE word unsigned;                         /* Self-relative byte offset to free storage
    SIZ_FREE word unsigned;                         /* Bytes left in CNF free storage
    SIZ_USED word unsigned;                         /* Number of bytes used for storing strings
    ID word unsigned;                               /* Database dependent indentification data
    "BOOLEAN" word unsigned;                        /* Storage for values of parameters of type "bit"
    FILL_1 word fill prefix CNFDEF tag $$;          /* Spare -- reserved for future use
    "MASK" longword unsigned dimension 3;           /*see CNR$C_MAX_INX definition
    constant "LENGTH" equals . prefix CNF$ tag K;   /* Structure size
    constant "LENGTH" equals . prefix CNF$ tag C;   /* Structure size
                                                    /*F        MASK,L,CNR$C_MAX_INX/32        /* One bit in mask for eac
                                                    /* The bit is set while the field is active.

end CNFDEF;
end_module $CNFDEF;
```

```
module $NDIDEF;
/*
/* REMOTE NODE INFORMATION

aggregate NDIDEF structure fill prefix NDI$;
    ADD word unsigned;
    CTI word unsigned;
    CTA longword unsigned;
    S_NNA longword unsigned;



    S_NLI longword unsigned;
    S_PUS longword unsigned;
    S_PAC longword unsigned;
    S_PPW longword unsigned;
    S_NUS longword unsigned;
    S_NAC longword unsigned;
    S_NPW longword unsigned;
    S_RPA longword unsigned;
    S_TPA longword unsigned;
    ACC byte unsigned;
    PRX byte unsigned;
    SNV byte unsigned;
    INB byte unsigned;



    FILL_2 byte fill prefix NDIDEF tag $$;
    SDV byte unsigned;
    CPU byte unsigned;
    STY byte unsigned;
    IHO word unsigned;
    OHO word unsigned;
    DAD longword unsigned;
    DCT longword unsigned;
    S_SLI longword unsigned;
    S_SPA longword unsigned;
    S_LOA longword unsigned;
    S_SLO longword unsigned;
    S_TLO longword unsigned;
    S_SID longword unsigned;
    S_DUM longword unsigned;
    S_SDU longword unsigned;
    S_DFL longword unsigned;
    S_HWA longword unsigned;
    constant "LENGTH" equals . prefix NDI$ tag K;
    constant "LENGTH" equals . prefix NDI$ tag C;


end NDIDEF;
end_module $NDIDEF;
```

```
/*        The following are commonly defined for all nodes

/* Node address - zero if NDI is for local node
/* Counter timer (units = sec)
/* Abslute due timer for counters to be logged
/* Name
/*
/*        The following are defined for some, but not most, nodes
/*
/* Line used if NDI is a "loopback" node
/* Priv user id
/* Priv account
/* Priv password
/* NonPriv user id
/* NonPriv account
/* NonPriv psw
/* Receive password
/* Transmit password
/* Access switch (inbound,outbound,etc)
/* Proxy access switch (inbound, outbound, etc)
/* System node version
/* Async Line - Inbound node type
/*
/*        The following are for nodes to be downline-loaded or uplin
/*
/* Spare used for alignment. Reserved for future use
/* Service device type
/* CPU type
/* Software type
/* Host address (input)
/* Host address (output)
/* Dump address
/* Dump count
/* Service line
/* Service password
/* Load file
/* Secondary loader
/* Tertiary loader
/* Software ID
/* Dump file
/* Secondary dumper
/* Diagnostic load file
/* NI hardware address for node
/* Structure size
/* Structure size
/* (used to down-line load a node which isn't
/*  up on the network yet)
```

```
module $LNIDEF;
/*
/* LOCAL NODE INFORMATION
/*
                                                        /*
                                                        /*        Define local node states
                                                        /*
constant STA_ON equals 0  prefix LNI tag $C;            /* Node available for general use
constant STA_OFF        equals 1  prefix LNI tag $C;    /* Node shutting down, no connects allowed
constant STA_SHUT       equals 2  prefix LNI tag $C;    /* No new connects allowed - shutting down
constant STA_RSTR       equals 3  prefix LNI tag $C;    /* Node available for outbound connects only
constant STA_INIT       equals 4  prefix LNI tag $C;    /* State used for ACP initialization
                                                        /*
                                                        /*        Define the CNF structure
                                                        /*

aggregate LNIDEF structure fill prefix LNI$;
    ADD word unsigned;                                  /* Node address
    STA byte unsigned;                                  /* State
    ETY byte unsigned;                                  /* Local node type

    MLK word unsigned;                                  /* Maximum links allowed
    MAD word unsigned;                                  /* Maximum node address
    MBU word unsigned;                                  /* Maximum transport buffers
    MCO word unsigned;                                  /* Maximum cost

    MHO byte unsigned;                                  /* Maximum hops
    MVI byte unsigned;                                  /* Maximum visits
    MLN byte unsigned;                                  /* Maximum circuits (used to be called lines)
    LPD byte unsigned;                                  /* Default LOOP data

    LPC word unsigned;                                  /* Default LOOP count
    LPL word unsigned;                                  /* Default LOOP length
    LPH byte unsigned;                                  /* Default LOOP help type
    FILL_1 byte fill prefix LNIDEF tag $$;              /* spare for alignment

    BUS word unsigned;                                  /* Transport forwarding buffer size
                                                        /* (maximum size that we will receive and forward)
    SBS word unsigned;                                  /* Transport segment buffer size
                                                        /* (maximum size that we will transmit)
    RSI word unsigned;                                  /* Routine supression interval (units = sec)
    IAT word unsigned;                                  /* Inactivity timer (units = sec)

    ITI word unsigned;                                  /* Incoming timer (units = sec)
    OTI word unsigned;                                  /* Outgoing timer (units = sec)
    RTI word unsigned;                                  /* Routing timer (units = sec)
    BRT word unsigned;                                  /* Broadcast routing timer (units = sec)

    MBE word unsigned;                                  /* Maximum broadcast endnodes
    MBR word unsigned;                                  /* Maximum broadcast routers

    DFA byte unsigned;                                  /* Delay factor
```

```
    DWE byte unsigned;                          /* Delay weight
    RFA byte unsigned;                          /* Retransmit factor
    DAC byte unsigned;                          /* Default access (inbound,outbound,etc)

    DPX byte unsigned;                          /* Default proxy access (inbound,outbound,etc)
    PIQ word unsigned;                          /* Pipeline quota
    FILL_2 byte fill prefix LNIDEF tag $$;      /* Spare used for alignment. Reserved for future use.

    SAD longword unsigned;                      /* X.25 sub-address range

    MAR byte unsigned;                          /* Maximum areas
    AMH byte unsigned;                          /* Area maximum hops
    AMC word unsigned;                          /* Area maximum cost
    ALA word unsigned;                          /* Alias local address (cluster node address)

    ALM word unsigned;                          /* Alias max links
    FILL_3 word fill prefix LNIDEF tag $$;      /* Spare for alignment

    S_NAM longword unsigned;                    /* Node name
    S_IDE longword unsigned;                    /* System identification
    S_NVE longword unsigned;                    /* NSP version
    S_RVE longword unsigned;                    /* Routing version
    S_MVE longword unsigned;                    /* Network Management version
    constant "LENGTH" equals . prefix LNI$ tag K;   /* Structure size
    constant "LENGTH" equals . prefix LNI$ tag C;   /* Structure size

end LNIDEF;

end_module $LNIDEF;
```

```
module $LLIDEF;
/*
/* LOGICAL LINK INFORMATION
/*
                                                        /*
                                                        /*        The following are commonly defined for all node counter bl
                                                        /*
/*
/* Node Counter Block  (NDC)
/*
/* The following data block is used to maintain statistics for each node in the
/* network.  A hash of these structures is contained in NETACP.
/*

aggregate NDCDEF structure fill prefix NDC$;
    ABS_TIM longword unsigned;                          /* Absolute time counter block was last zeroed
                                                        /*
                                                        /*      Network services layer counters
                                                        /*
    RSE word unsigned;                                  /* Trasmitted connect rejects due to resource errors
    RTO word unsigned;                                  /* Response timeouts
    CRC word unsigned;                                  /* Connects received
    CSN word unsigned;                                  /* Connects sent
    BRC longword unsigned;                              /* Bytes received
    BSN longword unsigned;                              /* Bytes sent
    PRC longword unsigned;                              /* Packets received
    PSN longword unsigned;                              /* Packets sent
    constant "LENGTH" equals . prefix NDC$ tag K;       /* Structure size
    constant "LENGTH" equals . prefix NDC$ tag C;       /* Structure size
end NDCDEF;

                                                        /*
                                                        /*        The following are commonly defined for all logical links
                                                        /*
aggregate LLIDEF structure fill prefix LLI$;
    XWB longword unsigned;                              /* Pointer to XWB
    LLN word unsigned;                                  /* Local link number
    PNA word unsigned;                                  /* Partner's node address
                                                        /*
                                                        /*      Network services layer Running Total counters
                                                        /*
    NDC_RT byte dimension NDC$C_LENGTH tag Z;           /* Running total counters
                                                        /*
                                                        /*      Network services layer Last Zeroed counters
                                                        /*
    NDC_LZ byte dimension NDC$C_LENGTH tag Z;           /* Last zeroed counters

    constant "LENGTH" equals . prefix LLI$ tag K;       /* Structure size
    constant "LENGTH" equals . prefix LLI$ tag C;       /* Structure size

end LLIDEF;
end_module $LLIDEF;
```

```
module $OBIDEF;
/*
/* NETWORK OBJECT INFORMATION
/*

                                                        /*        Define CNF storage
                                                        /*

aggregate OBIDEF structure fill prefix OBI$;
    NUM byte unsigned;                                  /* Object number
    PRX byte unsigned;                                  /* Proxy login switch (inbound, outbound, etc)
    CHN word unsigned;                                  /* Channel over which declaration occurred
    LPR longword unsigned;                              /* Low order privilege mask
    HPR longword unsigned;                              /* High order privilege mask
    UCB longword unsigned;                              /* Associated NET UCB if declared task
    PID longword unsigned;                              /* Associated process i.d. if declared task

    S_NAM longword unsigned;                            /* Name
    S_FID longword unsigned;                            /* File id
    S_USR longword unsigned;                            /* User id
    S_ACC longword unsigned;                            /* Account
    S_PSW longword unsigned;                            /* Password
    constant "LENGTH" equals . prefix OBI$ tag K;       /* Structure size
    constant "LENGTH" equals . prefix OBI$ tag C;       /* Structure size
end OBIDEF;
end_module $OBIDEF;
```

```
module $CRIDEF;
/*
/* CIRCUIT INFORMATION
/*



aggregate CRIDEF structure fill prefix CRI$;
    S_NAM longword unsigned;
    OWPID longword unsigned;
    CTA longword unsigned;



    STA byte unsigned;
    CHN byte unsigned;
    LCT word unsigned;

    S_LOO longword unsigned;
    HET word unsigned;
    FILL_1 word fill prefix CRIDEF tag $$;

    COS byte unsigned;
    MRC byte unsigned;
    RCT word unsigned;

    S_NUM longword unsigned;

    POL byte unsigned;
    PLS byte unsigned;
    TYP byte unsigned;
    FILL_2 byte fill prefix CRIDEF tag $$;

    S_DTE longword unsigned;

    MBL word unsigned;
    MWI byte unsigned;
    TRI byte unsigned;

    BBT word unsigned;
    TRT word unsigned;

    CHN word unsigned;
    USE byte unsigned;
    MRB byte unsigned;

    MTR byte unsigned;
    ACB byte unsigned;
    ACI byte unsigned;
    IAB byte unsigned;

    IAI byte unsigned;
/*
/*        Define CNF storage for fields used internally
/*

/* Circuit name
/* PID of temporary owner of line in service state
/* Absolute due time for counter logging
/*
/*        Define CNF storage for fields defined by the NICE protocol
/*
/* State
/* X.25 Channel No.
/* Counter timer

/* Loopback name
/* Hello timer
/* spare

/* Cost
/* Maximum recalls
/* Recall timer

/* Call Number

/* Polling state
/* Polling substate
/* Type
/* spare for alignment

/* DTE

/* Maximum block
/* Maximum window
/* Tributary

/* Babble timer
/* Transmit timer

/* X.25 channel
/* X.25 Usage
/* Maximum receive buffers

/* Maximum transmits
/* Active base
/* Active increment
/* Inactive base

/* Inactive increment
```

```
    IAT byte unsigned;                          /* Inactive threshold
    DYB byte unsigned;                          /* Dying base
    DYI byte unsigned;                          /* Dying increment

    DYT byte unsigned;                          /* Dying threshold
    DTH byte unsigned;                          /* Dead threshold
    XPT byte unsigned;                          /* Transport protocol
    MRT byte unsigned;                          /* Maximum routers on NI

    RPR byte unsigned;                          /* Router priority on NI
    VER byte unsigned;                          /* Async Line - Verification:
                                                /*   Enabled, Disabled, Inbound
    FILL_2 word fill prefix CRIDEF tag $$;      /* spare

    S_NET longword unsigned;                    /* X.25 network name

    LOOPNODE byte unsigned;                     /* Loopnode name length
    LOOPNODE byte dimension 6 tag T;            /* Loopnode name text
    FILL_3 byte fill prefix CRIDEF tag $$;      /* spare

    constant "LENGTH" equals . prefix CRI$ tag K;   /* Structure size
    constant "LENGTH" equals . prefix CRI$ tag C;   /* Structure size
end CRIDEF;

end_module $CRIDEF;
```

```
module $PLIDEF;
/*
/* PHYSICAL LINE INFORMATION
/*

aggregate PLIDEF structure fill prefix PLI$;
    S_NAM longword unsigned;                    /* Line name
    CTA longword unsigned;                      /* Absolute due time for counter logging
                                                /*
                                                /*      Define CNF storage for fields defined by the NICE protocol
    BFN byte unsigned;                          /* Number of buffers in receive pool
    STA byte unsigned;                          /* State
    SUB byte unsigned;                          /* Substate
    PRO byte unsigned;                          /* Protocol

    LCT word unsigned;                          /* Counter timer
    STI word unsigned;                          /* Service timer

    HTI word unsigned;                          /* Holdback timer
    RTT word unsigned;                          /* Retransmit timer

    MBL word unsigned;                          /* Maximum block
    MRT byte unsigned;                          /* Maximum retransmits
    MWI byte unsigned;                          /* Maximum window
    SLT word unsigned;                          /* Scheduling timer
    DDT word unsigned;                          /* Dead timer

    DLT word unsigned;                          /* Delay timer
    SRT word unsigned;                          /* Stream timer

    S_MCD longword unsigned;                    /* X.25 KMX microcode dump file [WRITE ONLY - ONE SHOT]

    S_HWA longword unsigned;                    /* NI hardware address [READ ONLY]
    EPT word unsigned;                          /* Ethernet protocol type
    MOD byte unsigned;                          /* X.25 mode (DTE, DCE, etc.)
    TPI byte unsigned;                          /* Transmit pipeline

    LNS longword unsigned;                      /* Async Line - Line speed

    SWI byte unsigned;                          /* Async Line - Switch
    HNG byte unsigned;                          /* Async Line - Hangup
    BFS word unsigned;                          /* Buffer size to override executor buffer size

    S_NET longword unsigned;                    /* X.25 network name

    constant "LENGTH" equals . prefix PLI$ tag K;       /* Structure size
    constant "LENGTH" equals . prefix PLI$ tag C;       /* Structure size
end PLIDEF;
end_module $PLIDEF;
```

```
module $EFIDEF;
/*
/* EVENT LOGGING FILTER INFORMATION
/*
                                                        /*
                                                        /*        Define the CNF structure
                                                        /*

aggregate EFIDEF structure fill prefix EFI$;
    SIN word unsigned;                                  /* Sink node address
    SP1 word unsigned;                                  /* Spare
    B1 longword unsigned;                               /* For user defined use
    B2 longword unsigned;                               /* For user defined use

    S_EVE longword unsigned;                            /* Event list
    S_SB1 longword unsigned;                            /* For user defined use
    S_SB2 longword unsigned;                            /* For user defined use
    S_SB3 longword unsigned;                            /* For user defined use
    constant "LENGTH" equals . prefix EFI$ tag K;       /* Structure size
    constant "LENGTH" equals . prefix EFI$ tag C;       /* Structure size

end EFIDEF;

end_module $EFIDEF;
```

```
module $ESIDEF;
/*
/* EVENT LOGGING SINK INFORMATION
/*
                                                        /*
                                                        /*        Define logging sink states
                                                        /*
constant STA_ON equals 0  prefix ESI tag $C;            /* Logging is on
constant STA_OFF        equals 1  prefix ESI tag $C;    /* Logging is off
constant STA_HLD        equals 2  prefix ESI tag $C;    /* Hold events
                                                        /*
                                                        /*   Define logging sink types
                                                        /*
constant SNK_CON        equals 1  prefix ESI tag $C;    /* Console
constant SNK_FIL        equals 2  prefix ESI tag $C;    /* File
constant SNK_MON        equals 3  prefix ESI tag $C;    /* Monitor
                                                        /*
                                                        /*        Define the CNF structure
                                                        /*

aggregate ESIDEF structure fill prefix ESI$;
    SNK byte unsigned;                                  /* Sink type
    STA byte unsigned;                                  /* Sink state
    SP1 word unsigned;                                  /* Spare
    B1 longword unsigned;                               /* For user defined use
    B2 longword unsigned;                               /* For user defined use

    S_LNA longword unsigned;                            /* Sink name
    S_SB1 longword unsigned;                            /* For user defined use
    S_SB2 longword unsigned;                            /* For user defined use
    S_SB3 longword unsigned;                            /* For user defined use
    constant "LENGTH" equals . prefix ESI$ tag K;       /* Structure size
    constant "LENGTH" equals . prefix ESI$ tag C;       /* Structure size
end ESIDEF;

end_module $ESIDEF;
```

```
module $SPIDEF;
/*
/* NETWORK SERVER PROCESS INFORMATION
/*
                                                        /*
                                                        /*        Define CNF storage
                                                        /*

aggregate SPIDEF structure fill prefix SPI$;
    PID longword unsigned;                              /* Server PID
    IRP longword unsigned;                              /* IRP of waiting DECLSERV QIO. 0 if process active
    RNA word unsigned;                                  /* Remote node address which initially started server
    CHN word unsigned;                                  /* Channel associated with L_IRP (waiting DECLSERV)

    S_ACS longword unsigned;                            /* ACS used initially to start server process
    S_RID longword unsigned;                            /* Remote user ID which initially started server
    S_SFI longword unsigned;                            /* Last (current) filespec given to server
    S_NCB longword unsigned;                            /* Last (current) NCB given to server
    S_PNM longword unsigned;                            /* Last (current) process name given to server
    constant "LENGTH" equals . prefix SPI$ tag K;       /* Structure size
    constant "LENGTH" equals . prefix SPI$ tag C;       /* Structure size
end SPIDEF;

end_module $SPIDEF;
```

```
module $WQEDEF;
/*
/*  Work Queue Elements (WQE) are used by the ACP to serialize and standardize
/*  all schedulable but non-IRP oriented work.  Datalink state transition
/*  control and events originating from ASTs are examples.
/*
/*  The WQE structure is depicted below.  The WQE$B_SUB field is used to
/*  determine if any special processing is needed when the WQE is queued or
/*  dequeued as follows:
/*

aggregate WQEDEF structure fill prefix WQE$;
    FLINK longword unsigned;                          /* Queue forward link
    BLINK longword unsigned;                          /* Queue backward link
    SIZE word unsigned;                               /* Bytes allocated for the WQE
    TYPE byte unsigned;                               /* Structure type code
    SUB byte unsigned;                                /* Structure sub-type code as follows:
                                                      /*
    constant SUB_BAS    equals 0  prefix WQE tag $C;  /* The WQE is the base of a list - NOTE low bit clear
                                                      /* for this SUB constant only!!!
                                                      /*
    constant SUB_ACP    equals 1  prefix WQE tag $C;  /* The WQE was spawned during normal internal ACP
                                                      /* activity, e.g., during IO$_ACPCONTROL Qio activity.
                                                      /* No special action is required when it is queued.
                                                      /* When it is dequeued, dispatch directly to the action
                                                      /* routine which responsible for deallocating it.
                                                      /*
    constant SUB_AST    equals 3  prefix WQE tag $C;  /* The WQE is the consequence of a miscellaneous AST,
                                                      /* e.g., a datalink Qio AST.  If its the first entry
                                                      /* queued then $WAKE the ACP.  When it is dequeued,
                                                      /* dispatch directly to the action routine -  which is
                                                      /* responsible for deallocating it.
                                                      /*
    constant SUB_MBX    equals 5  prefix WQE tag $C;  /* The WQE is the consequence of a mailbox read AST. If
                                                      /* it is the first element queued then $WAKE the ACP.
                                                      /* When it is dequeued, it is sent to the mailbox
                                                      /* servicing routine - which permenantly owns the WQE.
                                                      /*
    constant SUB_TIM    equals 7  prefix WQE tag $C;  /* The WQE is the consequence of a timer AST. If it is
                                                      /* the first element queued then $WAKE the ACP.  When
                                                      /* dequeued, another VMS timer must be set if there are
                                                      /* any more elements in the WQE timer queue.
    ACTION longword unsigned;                         /* Action routine address
    PM1_OVERLAY union fill;
        PM1 longword unsigned;                        /* Action routine first parameter
        PM1_FIELDS structure fill;
            EVT byte unsigned;                        /* Event code - interpreted in the context of the
                                                      /* QUAL field
            QUAL byte unsigned;                       /* REQIDT qualifier as follows:
                                                      /*
            constant QUAL_DLL   equals 1  prefix WQE tag $C;/* A data link event - REQIDT is the LPD$W_PTH value
            constant QUAL_RTG   equals 2  prefix WQE tag $C;/* An ACP routing event - REQIDT is always zero
```

```
            constant QUAL_CTM   equals 3  prefix WQE tag $C;/* A counter timer event - REQIDT identifies data base
            constant QUAL_ACT   equals 4  prefix WQE tag $C;/* The ACP active timer
            REQIDT word unsigned;                          /* Request identifier - interpreted in the context of
                                                           /* the QUAL field.
        end PM1_FIELDS;
    end PM1_OVERLAY;
    PM2 longword unsigned;                                 /* Action routine second parameter
    EVL_PKT longword unsigned;                             /* Ptr to the packet header if the WQE is being used
                                                           /* for event logging
    EVL_CODE word unsigned;                                /* Event logging code if WQE is used for event logging
    EVL_DT1 byte unsigned;                                 /* Event logging immediate data
    EVL_DT2 byte unsigned;                                 /* Event logging immediate data
    ADJ_INX word unsigned;                                 /* Adjacency to which this WQE applies (0 if none)
    SPARE word unsigned;                                   /* SPARE WORD
    constant "LENGTH" equals . prefix WQE$ tag K;          /* Structure size and begining of data area
    constant "LENGTH" equals . prefix WQE$ tag C;          /* Structure size and begining of data area
end WQEDEF;

end_module $WQEDEF;
```

```
module $DLLQIODEF;
/*
/* DLLQIO  - Datalink $QIO parameter block
/*

aggregate DLLQIODEF structure fill prefix DLLQIO$;
    FUNC longword unsigned;                          /* Function code
    P1 longword unsigned;                            /* QIO P1 parameter
    P2 longword unsigned;                            /* QIO P2 parameter
    P3 longword unsigned;                            /* QIO P3 parameter
    P4 longword unsigned;                            /* QIO P4 parameter
    P5 longword unsigned;                            /* QIO P5 parameter
    constant "LENGTH" equals . prefix DLLQIO$ tag K; /* Structure size
    constant "LENGTH" equals . prefix DLLQIO$ tag C; /* Structure size
end DLLQIODEF;

end_module $DLLQIODEF;
```

```
module $DEVTRNDEF;
/*
/* DEVTRN - Device translation table
/*

aggregate DEVTRNDEF structure fill prefix DEVTRN$;
    NETMAN byte unsigned;                                   /* Count of Network Management device name
    NETMAN character length 5;                              /* Network Management device name text
    VMS byte unsigned;                                      /* Count of VMS device name
    VMS character length 3;                                 /* VMS device name text
    DEV byte unsigned;                                      /* Device code
                                                            /* Define device codes
    constant(
          UNK                                               /* Unknown device
        , DMC                                               /* DMC-11
        , PCL                                               /* PCL-11
        , DMF                                               /* DMF-32
        , CI                                                /* CI-780
        , DMP                                               /* DMP-11
        , DUP                                               /* DUP-11 (for X.25)
        , KMS                                               /* KMC-11 (for X.25)
        , X25                                               /* X.25 datalink (datalink mapping)
        , UNA                                               /* DEUNA (Ethernet)
        , PPUNA                                             /* DEUNA operating in point-to-point mode
                                                            /* (internal testing purposes only!)
        , DMB                                               /* DMB-32
        ) equals 0 increment 1  prefix DEVTRN$C_D tag EV;

    PROT byte unsigned;                                     /* Default device protocol (NMA$C_LINPR...)
    CHAR_OVERLAY union fill;
        CHAR byte unsigned;                                 /* Device characteristics
        constant "LENGTH" equals . prefix DEVTRN$ tag K;/* Structure size
        constant "LENGTH" equals . prefix DEVTRN$ tag C;/* Structure size
        CHAR_BITS structure fill;
            MULTI bitfield mask;                            /* Multi-unit device
        end CHAR_BITS;
    end CHAR_OVERLAY;
end DEVTRNDEF;

end_module $DEVTRNDEF;
```

```
module $NDBDEF;
/*+
/* NDB - DEFINE OPCOM MESSAGE CODES
/*-

                                                        /* Message codes for OPCOM
constant(
      MSG_START                                         /* DECnet starting
    , MSG_SHUT                                          /* DECnet shutting down
    ) equals 1 increment 1  prefix NDB tag $C;

end_module $NDBDEF;
```

3 NETNPAGED SDL FILES

```
{
{               NETNPAGED.SDL   - DECnet non-paged control structures
{
{ Version:      'X-1'
{
{*****************************************************************************
{*                                                                           *
{*  COPYRIGHT (c) 1978, 1980, 1982, 1984, 1986 BY                            *
{*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                   *
{*  ALL RIGHTS RESERVED.                                                     *
{*                                                                           *
{*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED    *
{*  ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE           *
{*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER      *
{*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY    *
{*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY       *
{*  TRANSFERRED.                                                             *
{*                                                                           *
{*  THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE     *
{*  AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT         *
{*  CORPORATION.                                                             *
{*                                                                           *
{*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS      *
{*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                  *
{*                                                                           *
{*                                                                           *
{*****************************************************************************

{ MODIFIED BY:
{
{       X-1     PRB0376         Paul Beck       23-JUL-1985 16:42
{               Add support for cluster node names, as follows:
{                       Add RCP$L_PTR_ALIAS
{                       Add ICB$W_FLG and ICB$V_FLG_ALIAS
{                       Add AREG structure
{
{       V03-020 TMH0020         Tim Halvorsen   22-Apr-1984
{               Add ELECT_TIM flag to LPD for the election suppression timer.
{               Add PVC_ACCESSED flag to indicate PVC is ACCESSed.
{
{       V03-019 RNG0019         Rod Gamache     24-Mar-1984
{               Add NET$V_ACT bit to RCB$B_STATUS byte.
{
{       V03-018 PRB0319         Paul Beck       9-Mar-1984  9:34
{               Add NETUPD$_TEST_ADJ
{
{       V017    RNG0017         Rod Gamache     17-Feb-1984
{               Add NETUPD$_CRELNK for creating logical links.
{               Add ACT_TIMER and AQB_CNT bytes to the RCB.
{
{       V016    TMH0016         Tim Halvorsen   11-Jul-1983
```

```
{               Add executor cluster address to RCB.
{
{       V015    TMH0015         Tim Halvorsen   29-Mar-1983
{               Fix comments about MCOUNT offset
{
{       V014    TMH0014         Tim Halvorsen   13-Mar-1983
{               Add flag to disable AOA lookups (in the event we are
{               an isolated area router).
{               Add word/quadword datalink alignment flags.
{               Remove XMT_RESTR flag and add XMT_DALLY flag.
{
{       V013    TMH0013         Rod Gamache     25-Feb-1983
{               Add flag to toggle datalink line on circuit startup.
{
{       V012    TMH0012         Tim Halvorsen   14-Feb-1983
{               Add cell to hold datalink buffer size in LPD.
{               Remove XWB/LSB structures (now replaced by
{               a separate SDL).
{
{       V011    TMH0011         Tim Halvorsen   13-Oct-1982
{               Add storage for area routing support.
{
{       V010    TMH0010         Tim Halvorsen   28-Sep-1982
{               Add cell to RCB giving the designated router
{               if we are an endnode.
{
{       V009    RNG0001         Rod Gamache     20-Aug-1982
{               Add "broadcast to all endnodes" flag to LPD.
{
{       V008    TMH0008         Tim Halvorsen   07-Jul-1982
{               Add "broadcast circuit" flag to LPD.  Include "standard"
{               12 byte header.  Removed RCB$L_LOC_LPD and RCB$L_ECL_IRP.
{               Added LPD$C_LOC_INX.   Added RCB$L_PTR_JNX and RCB$L_PTR_DCS.
{               Add cost word to LPD.
{               Remove all fields in LPD, which are in new ADJ block.
{               Add RCB$W_MAX_RTG and MAX_ADJ.
{               Add LPD cell to point to a "most recently received election
{               message" for BC LPDs.
{               Add cell to RCB to hold the type of executor (routing,
{               endnode, etc.).
{               Add cell to LPD to hold the our NI router priority.
{               Add cell to LPD to hold the designated router on the circuit,
{               so that NETDRIVER can send the Router Hello msg to endnodes.
{               Add NETUPD message code to instruct NETDRIVER to send a hello
{               message immediately.
{               Add SRM bitmasks to LPD, in order to hold "send routing msg"
{               context for each circuit.  This is required for segmented
{               routing messages.  Remove RCB$V_STS_XRT flag, since that is
{               now replaced by the bitmasks.
{               Add cell to RCB to record current position in the ADJ vector
{               to be used by the listener process, since it will only do a
{               fraction of the adjacencies each second.
```

```
{               Add cell to LPD which determines the type of node we will
{               "look like" on that circuit (endnode, router, etc.)
{
{       V007    TMH0007         Tim Halvorsen   23-Jun-1982
{               Add CXB_FREE listhead to RCB
{
{       V006    TMH0006         Tim Halvorsen   19-May-1982
{               Add LPD$B_PVCFLG field and associated flags.
{               Add LPD$V_X25, LPD$V_X25BLK and LPD$V_INCOMING flags
{               Add LPD$B_STARTUPS to record number of startup
{               attempts since last "run" state
{               Add definitions for NI adjacency database.
{
{       V03-05  ADE0033         A.Eldridge      15-Feb-1982
{               Added RCB$B_ECL_RFLW and changed XWB$B_SPARE to XWB$B_INC_RFLW.
{
{       V03-04  ADE0032         A.Eldridge      15-Feb-1982
{               Added XWB$L_DI_CXB.
{
{       V03-03  ADE0031         A.Eldridge      18-Dec-1981
{               Added ICB$x_RID and XWB$x_RID.
{
{       V03-02  ADE0030         A.Eldridge      30-Nov-1981
{               Added RCB$B_ECL_DAC and RCB$B_ECL_DPX.
```

```
{
{ Routing Control Block definition.
{
{ The RCB is used as the network Volume Control Block.
{
{ Because miscellaneous software scans the I/O data base, the following fields
{ should be kept at the same offset as their VCB counter-parts:
{
{       RCB$W_TRANS     = VCB$W_TRANS    (offset = 12 decimal)
{       RCB$L_AQB       = VCB$L_AQB      (  "    = 16    "    )
{
module $RCBDEF;


aggregate RCBDEF structure fill prefix RCB$;
    IRP_FREE quadword unsigned;                 /* Listhead of free IRPs
    SIZE word unsigned;                         /* Bytes allocated for structure
    TYPE byte unsigned;                         /* Structure type
    STATUS_OVERLAY union fill;
        STATUS byte unsigned;                   /* Status flags
        STATUS_BITS structure fill;
            LVL2 bitfield mask;                 /* True if we can use level 2 routing (AOA, etc)
                                                /* False if we detect that we are an isolated area router
            ACT bitfield mask;                  /* (ROUTING LAYER BIT) Set if NETACP is considered active
        end STATUS_BITS;
    end STATUS_OVERLAY;
    TRANS word unsigned;                        /* Outstanding transaction count
    ADDR word unsigned;                         /* Local node address

    AQB longword unsigned;                      /* Ptr to AQB
    ACP_UCB longword unsigned;                  /* Ptr to Network ACP's UCB
    PTR_JNX longword unsigned;                  /* Ptr to journal buffer
    PTR_OA longword unsigned;                   /* Ptr to Output Adjacency vector
    PTR_AOA longword unsigned;                  /* Ptr to Area Output Adjacency vector

    PTR_LTB longword unsigned;                  /* Ptr to logical link table (dispatch vector)
    PTR_LPD longword unsigned;                  /* Ptr to Logical Path Descriptor vector
    PTR_ADJ longword unsigned;                  /* Pointer to vector of adjacency blocks (0 based)
    PTR_TQE longword unsigned;                  /* Ptr to internal Timer Queue element
    PTR_NDC longword unsigned;                  /* Ptr to Node Counters vector

    PTR_DCS longword unsigned;                  /* Ptr to NETDRIVER's control storage
    PTR_AREG longword unsigned;                 /* Ptr to NETDRIVER's alias registration table
    LOC_RCV quadword unsigned;                  /* Receive listhead for "local" Datalink
    LOC_XMT quadword unsigned;                  /* Transmit listhead for "local" Datalink
    IRP_WAIT quadword unsigned;                 /* Listhead of fork processes waiting for an IRP

    MCOUNT word unsigned;                       /* Mount count - includes current logical links plus 1
                                                /* for the ACP reference
    CUR_LLNK word unsigned;                     /* Current number of local logical links
    MAX_LNK word unsigned;                      /* Max allowable logical links
    MAX_ADDR word unsigned;                     /* Max allowable node address
```

```
MAX_LPD byte unsigned;                  /* Max number of DLL LPDs
MAX_SNK byte unsigned;                  /* Max IRPs queueable to any sink queue
MAX_VISIT byte unsigned;                /* Max nodes a packet may visit
INT_PTH byte unsigned;                  /* Index of path to intercept node: 0=>no intercept

ACT_DLL byte unsigned;                  /* Active data links
STI byte unsigned;                      /* Internal state
ECL_RFLW byte unsigned;                 /* ECL receiver pipeline quota (in packets)
ECL_RFA byte unsigned;                  /* ECL retransmit factor

ECL_DFA byte unsigned;                  /* ECL delay factor
ECL_DWE byte unsigned;                  /* ECL delay weight
ECL_DAC byte unsigned;                  /* ECL default access state
ECL_DPX byte unsigned;                  /* ECL default proxy access state

MAX_ADJ word unsigned;                  /* Size of adjacency vector (PTR_ADJ), in entries
MAX_RTG word unsigned;                  /* Number of "routing destinations" (LPDs + BRAs)
DLE_XWB longword unsigned;              /* Linked list head for direct-circuit-access XWBs

TIM_RSI word unsigned;                  /* Min delay before next routing update in seconds
TIM_RTI word unsigned;                  /* Max time before next routing update in seconds

TIM_IAT word unsigned;                  /* Max logical link inactivity interval in seconds
TIM_CNI word unsigned;                  /* Max inbound connect interval in seconds
TIM_CNO word unsigned;                  /* Max outbound connect interval in seconds
TIM_CTI word unsigned;                  /* Sample interval in seconds

ECLSEGSIZ word unsigned;                /* Default ECL data segment size (excludes header size)
TOTBUFSIZ word unsigned;                /* Buffer size including NSP header + 6 bytes
                                        /* + CXB$C_OVERHEAD bytes for datalink overhead
CUR_PKT word unsigned;                  /* Current total IRPs
MAX_PKT word unsigned;                  /* Max total IRPs

PKT_FREE word unsigned;                 /* Packets left in Transmit pool
PKT_PEAK word unsigned;                 /* Peak number of packets used

PKT_FAIL byte unsigned;                 /* Packet allocation failures (over packet quota)
MEM_FAIL byte unsigned;                 /* Packet allocation failures (insufficent memory)
ETV byte unsigned;                      /* Type of executor node (ADJ$C_PTY_xxx)
HOMEAREA byte unsigned;                 /* Our home area number (only if area router)

MAX_AREA byte unsigned;                 /* Max allowable area address
ACT_TIMER byte unsigned;                /* ACP activity timer
"ALIAS" word unsigned;                  /* Alias VAXcluster address (0 if none)

MAX_ALNK word unsigned;                 /* Max allowable links using cluster alias
MAX_LLNK word unsigned;                 /* Max allowable links using local node
CUR_ALNK word unsigned;                 /* Current number of alias links
                                        /*
                                        /*        Transport layer counters
                                        /*
```

```
    ABS_TIM longword unsigned;                          /* Absolute time counters were last zeroed
    CNT_NOL_OVERLAY union fill;
        CNT_NOL byte unsigned;                          /* Node out-of-range packet loss
        CNT_1ST byte unsigned;                          /* First counter cell marker
        constant CNT_SIZE       equals 12  prefix RCB tag $C;/* Number of bytes used for counters
    end CNT_NOL_OVERLAY;
    CNT_APL byte unsigned;                              /* Aged packet loss
    CNT_OPL byte unsigned;                              /* Oversized packet loss
    CNT_PFE byte unsigned;                              /* Packet format error

    CNT_RUL byte unsigned;                              /* Partial routing update loss
    CNT_VER byte unsigned;                              /* Verification rejects
    CNT_NUL word unsigned;                              /* Node unreachable packet loss
    CNT_XRE word unsigned;                              /* Xmitted connect resource errors (ECL layer counter)
    CNT_MLL word unsigned;                              /* Maximum logical links active (ECL layer counter)
                                                        /*
                                                        /*        End of transport counters
                                                        /*

    CXB_FREE quadword unsigned;                         /* Free CXB listhead
    LSN_ADJ byte unsigned;                              /* Current position in ADJ for listener processing
    AQB_CNT byte unsigned;                              /* Count of entries on AQB
    DRT word unsigned;                                  /* Designated router ADJ index for all transmits
                                                        /* with an unspecified output adjacency
    LVL2 word unsigned;                                 /* Level 2 router ADJ index for all transmits with
    constant "LENGTH" equals . prefix RCB$ tag K;       /* Structure size
    constant "LENGTH" equals . prefix RCB$ tag C;       /* Structure size
                                                        /* an unspecified output adjacency (ETY=PH4 only)
end RCBDEF;

end_module $RCBDEF;
```

```
module $LPDDEF;
/*
/* Logical Path Descriptor  (LPD)
/*
/* The following control block describes a path to a data sink/source -- either
/* a datalink driver (e.g., XMDRIVER) or an end communications level (ECL)
/* driver (e.g., NETDRIVER).
/*

aggregate LPDDEF structure fill prefix LPD$;
    REQ_WAIT quadword unsigned;                         /* Listhead of fork processes waiting for a 'request'
                                                        /* slot on the LPD sink queue.
    SIZE word unsigned;                                 /* Structure size
    TYPE byte unsigned;                                 /* Structure type
    STARTUPS byte unsigned;                             /* Number of datalink startup attempts since last "run"
    WIND longword unsigned;                             /* Driver context - WIND field image for IRPs
    UCB longword unsigned;                              /* Driver context - UCB address

    CHAN word unsigned;                                 /* ACP channel to device
    TIM_TLK word unsigned;                              /* "Talker" timer
    INT_TLK word unsigned;                              /* "Talker" interval (used to init TIM_TLK)
    TSTCNT byte unsigned;                               /* Number of test messages left to send before entering
                                                        /* the RUN state
    ASTCNT byte unsigned;                               /* Number of outstanding ASTs
    IRPCNT byte unsigned;                               /* Number of outstanding IRPs queued by NETDRIVER
    ETY byte unsigned;                                  /* Our node type, on this circuit
    XMT_SRL byte unsigned;                              /* Output "square root limiter" value
    XMT_IPL byte unsigned;                              /* Output queue "input packet limiter"
    PTH_OVERLAY union fill;
        PTH word unsigned;                              /* Path ID
        PTH_FIELDS structure fill;
            PTH_INX byte unsigned;                      /* Path index
            PTH_SEQ byte unsigned;                      /* Path sequence
            constant LOC_INX    equals 1  prefix LPD tag $C;/* Index associate with the primary "local" LPD
                                                        /*
                                                        /*        Status on control info
                                                        /*

        end PTH_FIELDS;
    end PTH_OVERLAY;
    STS_OVERLAY union fill;
        STS word unsigned;                              /* Status bits as follows:
        STS_BITS structure fill;
            ACTIVE bitfield mask;                       /* Path is active
            STRTIM bitfield mask;                       /* Set if restart supression timer is ticking
            DLE bitfield mask;                          /* Set if in use for physical line service
            ACCESS bitfield mask;                       /* Set if LPD is being "accessed" by a server process
            RUN bitfield mask;                          /* Set if active and in use for normal data msgs
            XBF bitfield mask;                          /* Set if Xmitter uses buffered I/O
            RBF bitfield mask;                          /* Set if Receiver uses buffered I/O
            X25 bitfield mask;                          /* Set if X.25 datalink mapping used on this circuit
            X25BLK bitfield mask;                       /* Set if blocking requested for X.25 datalink
            INCOMING bitfield mask;                     /* Set if X.25 circuit waiting for incoming call
```

```
        BC bitfield mask;                       /* Set if circuit is a broadcast circuit (NI)
        XEND bitfield mask;                     /* Set if "send hello msg to all endnodes" is requested
        TOGGLE bitfield mask;                   /* Set if listener timeout on DMC line (toggles line)
        ALIGNW bitfield mask;                   /* Set if datalink requires word alignment
        ALIGNQ bitfield mask;                   /* Set if datalink requires quadword alignment
        ELECT_TIM bitfield mask;                /* Set if election suppression timer ticking
    end STS_BITS;
end STS_OVERLAY;
XMTFLG_OVERLAY union fill;
    XMTFLG byte unsigned;

    XMTFLG_BITS structure fill;                 /* Xmit flags -- since a FFS is used to schedule message
        XMT_DALLY bitfield mask;                /* transimission, the order of these flags are crucial
        XMT_STR bitfield mask;                  /* Dally before sending a start msg (must preceed STR)
        XMT_VRF bitfield mask;                  /* Xmit a Transport start msg
        XMT_IDLE bitfield mask;                 /* Xmit a Transport verification msg
        XMT_RT bitfield mask;                   /* Signals "no more Transport init messages to send"
        XMT_HEL bitfield mask;                  /* Xmit a Transport routing message
        XMT_ART bitfield mask;                  /* Xmit a Transport hello message
    end XMTFLG_BITS;                            /* Xmit a Transport Area routing message
end XMTFLG_OVERLAY;
PVCFLG_OVERLAY union fill;
    PVCFLG byte unsigned;

    PVCFLG_BITS structure fill;                 /* X.25 PVC startup flags -- since a FFS is used to
        PVC_ACCESS bitfield mask;               /* schedule it, the order of these flags are crucial
        PVC_RESTRT bitfield mask;               /* Issue an IO$_ACCESS to establish the connection
        PVC_RESET bitfield mask;                /* Issue a "restart confirmation"
        FILL_2 bitfield length 4 mask;          /* Issue a "reset" or "reset confirmation"
        PVC_ACCESSED bitfield mask;             /* (reserve low 4 bits for state, 3 more bits unused)
    end PVCFLG_BITS;                            /* True if PVC ACCESSed
end PVCFLG_OVERLAY;
STI byte unsigned;                              /* Internal state used by the ACP for initialization
SUB_STA byte unsigned;                          /* Circuit sub-state
PLVEC byte unsigned;                            /* Associate PLVEC index
COST byte unsigned;                             /* Circuit cost
BCPRI byte unsigned;                            /* Circuit NI router priority
FILL_1 byte fill prefix LPDDEF tag $$;          /* spare
DRT word unsigned;                              /* Designated router on NI
RTR_LIST longword unsigned;                     /* For BC LPDs, address of "most recently received
                                                /* election message" from Router Hello messages
                                                /* (stored as an byte-counted string)
RCV_IRP longword unsigned;                      /* Address of suspended receiver IRP
                                                /*
                                                /*        Transport layer counters
                                                /*
ABS_TIM longword unsigned;                      /* Absolute time counters were last zeroed
CNT_APR_OVERLAY union fill;
    CNT_APR longword unsigned;                  /* Arriving packets received
    CNT_1ST byte unsigned;                      /* First counter cell marker
end CNT_APR_OVERLAY;
CNT_TPR longword unsigned;                      /* Transit packets received
```

```
    CNT_DPS longword unsigned;                      /* Departing packets sent
    CNT_TPS longword unsigned;                      /* Transit packets sent
    CNT_ACL word unsigned;                          /* Arriving congestion loss
    CNT_TCL word unsigned;                          /* Transit congestion loss
    CNT_LDN byte unsigned;                          /* Line down events
    CNT_IFL byte unsigned;                          /* Initialization failures
    constant CNT_SIZE   equals 22  prefix LPD tag $C;   /* Number of bytes used for counters
    BUFSIZ word unsigned;                           /* Datalink buffer size including Transport overhead
                                                    /* (variable route header depending on the datalink)
    SRM_POS byte unsigned;                          /* Current position in transmit XMT_SRM bitmask (bit !)
    SRM_LEFT byte unsigned;                         /* Number of transmit XMT_SRM bits left to check/process
    ASRM_POS byte unsigned;                         /* Current position in transmit XMT_ASRM bitmask (bit !)
    ASRM_LEFT byte unsigned;                        /* Number of transmit XMT_ASRM bits left to check/process

    SRM byte unsigned tag G dimension 4;            /*(32/8) ; "Send routing message" flags (one per segment)
    XMT_SRM byte unsigned tag G dimension 4;        /* Copy of SRM flags used for rtg msg transmission
    constant SRM_NODES  equals 32  prefix LPD tag $C;   /* Number of nodes per segment(bit)
    constant SRM_SHFT   equals 5  prefix LPD tag $C;    /* Bit shift for node<->bit!
    constant SRM_SIZE   equals 32  prefix LPD tag $C;   /* Number of bits in SRM bitmask (allows for 1024 nodes)

    ASRM byte unsigned tag G dimension 4;           /*(1/8); "Send area routing message" flags (one per segment)
    XMT_ASRM byte unsigned tag G dimension 4;       /* Copy of ASRM flags used for rtg msg transmission
    constant ASRM_AREAS equals 64  prefix LPD tag $C;   /* Number of areas per segment(bit)
    constant ASRM_SHFT  equals 6  prefix LPD tag $C;    /* Bit shift for area<->bit!
    constant ASRM_SIZE  equals 1  prefix LPD tag $C;    /* Number of bits in ASRM bitmask (allows for 64 areas)

    CACHE longword unsigned;                        /* Address of endnode cache storage (endnodes only)
    constant "LENGTH" equals . prefix LPD$ tag K;   /* Structure size
    constant "LENGTH" equals . prefix LPD$ tag C;   /* Structure size
end LPDDEF;

end_module $LPDDEF;
```

```
module $ADJDEF;
/*
/* Adjacency Node Data Base Block (ADJ)
/*
/* This block describes the contents of the Adjacency Node Data Base. The
/* Adjacency Node Data Base is used in conjunction with the LPD data base
/* to describe the destination to any node in the network.
/*

aggregate ADJDEF structure fill prefix ADJ$;
    STS_OVERLAY union fill;
        STS byte unsigned;                              /* Status flags
        STS_BITS structure fill;
            INUSE bitfield mask;                        /* Adjacency block in use
            RUN bitfield mask;                          /* Adjacency is up (can transmit pkts over it)
            RTG bitfield mask;                          /* Partner is a routing node (PH3,PH4-1,PH4-2)
            LSN bitfield mask;                          /* Listen timer is ticking
        end STS_BITS;
    end STS_OVERLAY;
    PTYPE byte unsigned;                                /* Type of partner node (routing, non-routing, etc.)
    constant PTY_UNK    equals -1  prefix ADJ tag $C;   /* Node type is unknown
    constant PTY_PH3    equals 0  prefix ADJ tag $C;    /* Phase III full routing
    constant PTY_PH3N   equals 1  prefix ADJ tag $C;    /* Phase III non routing
    constant PTY_PH2    equals 2  prefix ADJ tag $C;    /* Phase II
    constant PTY_AREA   equals 3  prefix ADJ tag $C;    /* Phase IV area routing
    constant PTY_PH4    equals 4  prefix ADJ tag $C;    /* Phase IV routing
    constant PTY_PH4N   equals 5  prefix ADJ tag $C;    /* Phase IV non routing
    LPD_OVERLAY union fill;
        LPD word unsigned;                              /* Path ID
        LPD_FIELDS structure fill;
            LPD_INX byte unsigned;                      /* Path index
            LPD_SEQ byte unsigned;                      /* Path sequence
        end LPD_FIELDS;
    end LPD_OVERLAY;
    PNA word unsigned;                                  /* Partner node address
    BUFSIZ word unsigned;                               /* Neighbor's block size
    INT_LSN word unsigned;                              /* Listener interval (computed from neighbor's hello)
    TIM_LSN word unsigned;                              /* Listener timer (seconds left)
    BCPRI byte unsigned;                                /* Broadcast priority
    constant "LENGTH" equals . prefix ADJ$ tag K;       /* Structure size
    constant "LENGTH" equals . prefix ADJ$ tag C;       /* Structure size
end ADJDEF;

end_module $ADJDEF;
```

```
module $XMCDEF;
/*+
/*
/* DMC counter block - provides offsets for SHOW QIO return of counters
/*
/*-


aggregate XMCDEF structure fill prefix XMC$;
    XMTBYTCNT longword unsigned;                    /* No. of bytes transmitted
    RCVBYTCNT longword unsigned;                    /* No. of bytes received
    XMTMSGCNT word unsigned;                        /* No. of msgs transmitted
    RCVMSGCNT word unsigned;                        /* No. of msgs received
    RCVNOBUF word unsigned;                         /* No. of "no buffer" NAKS received
    RCVHDRBCC word unsigned;                        /* No. of "header BCC error" NAKS received
    RCVDATBCC word unsigned;                        /* No. of "data BCC error" NAKS received
    XMTNOBUF word unsigned;                         /* No. of "no buffer" NAKS xmitted
    XMTHDRBCC word unsigned;                        /* No. of "header BCC error" NAKS xmitted
    XMTDATBCC word unsigned;                        /* No. of "data BCC error" NAKS xmitted
    XMTREPS word unsigned;                          /* No. of REPS xmitted
    RCVREPS word unsigned;                          /* No. of REPS received
    constant "LENGTH" equals . prefix XMC$ tag K;   /* Length of counter block
    constant "LENGTH" equals . prefix XMC$ tag C;   /* Length of counter block
end XMCDEF;

end_module $XMCDEF;
```

```
module $LTBDEF;
/*
/* LINK TABLE        - LTB
/*
/* This structure is maintained by NSP (NETDRIVER).  It contains all
/* local 'end communications layer' parameters and a vector of logical
/* link slots.
/*
aggregate LTBDEF structure fill prefix LTB$;
    SLT_NXT longword unsigned;                          /* Pointer into the link slot vector of the
                                                        /* slot candidate to be tried the next time
                                                        /* a link slot needs to be allocated
    ASLT_NXT longword unsigned;                         /* Next slot for cluster link
    SLT_TOT word unsigned;                              /* Total slots in vector
    SLT_LMT word unsigned;                              /* Total useable slots in vector
    ASLT_LMT word unsigned;                             /* Total useable cluster slots in vector
    LSLT_TOT word unsigned;                             /* Total slots for local links
    ASLT_TOT word unsigned;                             /* Total slots for cluster links
    SIZE word unsigned;                                 /* Size, in bytes, of this structure
    TYPE byte unsigned;                                 /* Structure identifier
    SPARE_1 byte unsigned;                              /* Reserved for future use
    SPARE_2 word unsigned;                              /* Reserved for future use
    XWB longword unsigned;                              /* XWB listhead
    PTR_ALIAS_SLOTS longword unsigned;                  /* Addr of the beginning of the alias link slot vector.
    SLOTS longword unsigned;                            /* The beginning of the logical link slot vector.
    constant "LENGTH" equals . prefix LTB$ tag K;       /* Structure size
    constant "LENGTH" equals . prefix LTB$ tag C;       /* Structure size
                                                        /* This is the first slot, not a pointer to it.
                                                        /* Each slot in the vector is 4 bytes.  If the
                                                        /* low bit is set then the slot is available and
                                                        /* the last used link ID for this slot can be
                                                        /* found in the high order word.  If the low bit
                                                        /* is clear then the slot contains a pointer to
                                                        /* a structure containing the link context and
                                                        /* state information.
                                                        /*
                                                        /* This vector is terminated by a longword of all
                                                        /* ones followed by a longword of all zeroes.
end LTBDEF;

end_module $LTBDEF;
```

```
module $ICBDEF;
/*
/* INTERNAL CONNECT BLOCK  - used to pass generic connect information
/*                           between the Network ACP and an End Communi-
/*                           cations Layer (ECL) driver (e.g. NETDRIVER)
/*
aggregate ICBDEF structure fill prefix ICB$;
    PATH word unsigned;                                 /* Path number (logical line) over which connect
                                                        /* was received or is to be transmitted.  It
                                                        /* contains a sequence number in the high byte.
                                                        /* A value of zero indicates no path specified.
    LOCLNK word unsigned;                               /* Local link address
    TIM_OCON word unsigned;                             /* Outbound connect timer, in seconds
    TIM_INACT word unsigned;                            /* Maximum inactivity timer, in seconds
    SIZE word unsigned;                                 /* Bytes allocated for structure
    TYPE byte unsigned;                                 /* Contains structure type code
    FILL_1 byte fill prefix ICBDEF tag $$;              /* (spare)
    RETRAN word unsigned;                               /* Maximum consecutive retransmissions before the
                                                        /* link is forcibly disconnected
    DLY_FACT word unsigned;                             /* Retransmission timer delay factor
    DLY_WGHT word unsigned;                             /* Retransmission timer delay weight
    SEGSIZ word unsigned;                               /* Maximum receive data segment size, in bytes,
                                                        /* excluding End Communications Layer protocol.
    constant LPRNAM     equals 20  prefix ICB tag $C;   /* Maximum length of counted process name
    LPRNAM byte unsigned;                               /* Count of bytes used in next field
    LPRNAM character length 19;                         /* Local process name
    constant RPRNAM     equals 20  prefix ICB tag $C;   /* Maximun length of counted process name
    RPRNAM byte unsigned;                               /* Count of bytes used in next field
    RPRNAM_OVERLAY union fill;
        RPRNAM character length 19;                     /* Remote process name
                                                        /*
                                                        /* Process names contain a type field in the
                                                        /* first byte, and have the following formats:
                                                        /*
                                                        /* <0> followed by 1 byte non-zero object type
                                                        /* <1> followed by an object type followed by
                                                        /*     a counted (16 text bytes,max) object name
                                                        /* <2> followed by an object type followed by
                                                        /*     2 byte group code, 2 byte user code, and
                                                        /*     a counted (12 text bytes,max) object name
                                                        /*
        RPRNAM_FIELDS structure fill;
            DSTFMT byte unsigned;                       /* Format byte of remote process name
            DSTOBJ byte unsigned;                       /* Object type byte of remote process name
            DSTDSC character;                           /* Destination descriptor part of remote proc name
            constant ACCESS     equals 64  prefix ICB tag $C;/* Maximum combined composite access string length
        end RPRNAM_FIELDS;
    end RPRNAM_OVERLAY;
    ACCESS byte unsigned;                               /* Count of bytes in following field
    ACCESS character length 63;                         /* Concatenated access control strings, with subfield
                                                        /* maximum lengths as follows (note that the sum of
                                                        /* the actual subfield lengths must be less than 63):
```

```
/*    F         USER,B                  /* Count of bytes used in next field
/*    F         USER,T,39          /* "User" field of account to be connected to
/*    F         PSW,B                   /* Count of bytes used in next field
/*    F         PSW,T,39          /* "Password" field of that account
/*    F         ACCT,B                  /* Count of bytes used in next field
/*    F         ACCT,T,39          /* "Account" field of that account

    DATA byte unsigned;                                 /* Count of bytes used in next field
    DATA character length 16;                           /* Optional connect data
    REMNOD word unsigned;                               /* Network address of partner
{     FILL_2 word fill prefix ICBDEF tag $$;            /* Spare for alignment
    STS structure tag W;                                /* Flags for NETDRIVER from NETACP
        STS_ALIAS bitfield mask;                        /* Connection originates from alias address
        end STS;
    FILL_3 byte fill prefix ICBDEF tag $$;              /* Spare for alignment
    RID byte unsigned;                                  /* Remote user (process, task, etc.) i.d.
    RID character length 16;                            /*
    constant "LENGTH" equals . prefix ICB$ tag K;       /* Structure size
    constant "LENGTH" equals . prefix ICB$ tag C;       /* Structure size
    constant RID        equals 16  prefix ICB tag $C;   /* Max size of RID text field
end ICBDEF;

end_module $ICBDEF;
```

```
module $NETUPDDEF;
/*+
/*
/* Function codes for XWB update routine (ACP communication routine)
/*
/*-
                                                        /* Code
constant(
      ABORT                                             /* Breaking link
    , CONNECT                                           /* Give NCB to task w/ declared name or object
    , EXIT                                              /* Task is exiting
    , PROCRE                                            /* Starting a new process
    , DLL_ON                                            /* Datalink starting for normal use
    , DLL_DLE                                           /* Datalink starting for service functions
    , CRELNK                                            /* Create a logical link control block
    , ABOLNK                                            /* Abort all links - network shutdown
    , DSCLNK                                            /* Disconnect the specified link
    , BRDCST                                            /* Broadcast mailbox message
    , REPLY                                             /* Send mailbox message to a specific mailbox
    , REACT_RCV                                         /* Reactivate datalink receiver
    , SEND_HELLO                                        /* Send hello message immediately
    , GET_ADJ                                           /* Get output ADJ for a given node address
    , TEST_ADJ                                          /* Test if given node is adjacent to endnode
    ) equals 1 increment 1  prefix NETUPD tag $;

end_module $NETUPDDEF;

module $NETMSGDEF;

/*+
/* Define event codes used to pass blocks to the ACP
/*
/*-

                                                        /*
constant(
      UNK                                               /* Unknown message
    , ILL                                               /* Illegal message
    , TR                                                /* Transport control message
    , IRP                                               /* IRP from datalink which is shutting down
    , APL                                               /* Aged packet
    , NUL                                               /* Node unreachable packet
    , NOL                                               /* Node out of range packet
    , PFE                                               /* Packet format error
    , LSN                                               /* Listner timer expired
    , OPL                                               /* Oversized packet loss
    , CRD                                               /* Circuit run down
    , ADJ                                               /* Adjacency up
    ) equals 1 increment 1  prefix NETMSG tag $C;

    end_module $NETMSGDEF;
```

```
module  $AREGDEF;
/*
/*      Alias REGistration block.
/*
/*      Contains information about a node in a VAXcluster which is
/*      participating in an alias (cluster node address). Pointed to by
/*      RCB$L_PTR_AREG and contains one entry (as described below)
/*      per available slot (as many slots as nodes may participate),
/*      following a standard 12-byte structure header.
/*
aggregate AREGDEF structure fill prefix AREG$;
    ADDR word unsigned;                 /* Local node address of registrant
    FLAGS_OVERLAY union fill;
        FLAGS byte unsigned;            /* Flag bits as follows:
        FLAGS_BITS structure fill;
            VERSION bitfield length 4;  /* Protocol version
            INCOMING bitfield mask;     /* Set if accepting incoming links
            ROUTER bitfield mask;       /* Set if node is a routing node
        end FLAGS_BITS;
    end FLAGS_OVERLAY;
    constant VERSION equals 0 tag C;    /* Current version number
    PRVINDEX byte unsigned;             /* Previous index used for CI
                                        /*   This entry is related to the source
                                        /*   link ID of an inbound CI which is
                                        /*   used to construct the index to this
                                        /*   entry. PRVINDEX contains the index
                                        /*   of the node to which the CI was
                                        /*   forwarded.
    MAXLINKS word unsigned;             /* Max # links for alias address
    CURLINKS word unsigned;             /* Current # links for alias address
    constant "LENGTH" equals . prefix AREG$ tag K;      /* Structure size
    constant "LENGTH" equals . prefix AREG$ tag C;      /* Structure size

end $AREGDEF;

end_module $AREGDEF;
```

4 NETUSR SDL FILES

```
{
{ NETUSR.SDL - system definitions for NETWORK ACP interface
{
{ Version:      'X-8'
{
{*****************************************************************************
{*                                                                           *
{*  COPYRIGHT (c) 1978, 1980, 1982, 1984, 1986 BY                            *
{*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                   *
{*  ALL RIGHTS RESERVED.                                                     *
{*                                                                           *
{*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED    *
{*  ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE           *
{*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER      *
{*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY    *
{*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY       *
{*  TRANSFERRED.                                                             *
{*                                                                           *
{*  THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE     *
{*  AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT         *
{*  CORPORATION.                                                             *
{*                                                                           *
{*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS      *
{*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                  *
{*                                                                           *
{*                                                                           *
{*****************************************************************************

{++
{
{ FACILITY:  VAX/VMS System Macro Libraries
{
{ ABSTRACT:
{
{       This file contains the SDL source for NETWORK control blocks.
{
{ ENVIRONMENT:
{
{       n/a
{
{ AUTHOR:  The VMS Group          CREATION DATE:  1-Aug-1976
{
{ MODIFIED BY:
{
{       X-8     PRB00X8         Paul Beck       17-OCT-1985 19:34
{               Add definition for Object Alias Incoming parameter
{
{       V04-004 BAS             Barry A. Scott  19-Sep-1985
{               Address XD5/9_IDTE parameter.
{
{       V04-003 BAS             Barry A. Scott  13-May-1985
```

```
{               Add X.25 Phase IV+ parameters.
{
{       V04-002 PRB00363        Paul Beck       31-MAR-1985 18:48
{               Supply definition for Transmit Pipeline
{
{       V04-001 PRB0360         Paul Beck       24-MAR-1985 17:54
{               Add asynchronous line support
{
{       V024    RNG0024         Rod Gamache             28-Feb-1984
{               Add IPID field for LLI database.
{
{       V023    MKP0001         Kathy Perko             06-Dec-1983
{               Add Node parameter, SERVICE NODE VERSION, for down line
{               load.
{
{       V022    TMH0022         Tim Halvorsen           04-Jul-1983
{               Add XAI database for X.25 gateway support.
{               Add LNI ALIAS (cluster node name) parameter.
{
{       V021    TMH0021         Tim Halvorsen           20-Apr-1983
{               Add Service (DLE) database.
{
{       V020    RNG0020         Rod Gamache             18-Apr-1983
{               Reserve some more 'database' codes for PSI.
{
{       V019    TMH0019         Tim Halvorsen           04-Mar-1983
{               Add DEVNAM circuit parameter (action routine).
{
{       V018    TMH0018         Tim Halvorsen           14-Feb-1983
{               Add line buffer size parameter.
{
{       V017    TMH0017         Tim Halvorsen           07-Dec-1982
{               Add "Next node to destination" node name, as well as
{               the node number which is already a parameter.
{               Add Ethernet protocol type parameter, so that Phase IV
{               can be used with another protocol type on the NI.
{
{       V016    TMH0016         Tim Halvorsen           05-Nov-1982
{               Add area database.
{
{       V015    TMH0015         Tim Halvorsen           23-Sep-1982
{               Add adjacency database.
{
{       V014    TMH0014         Tim Halvorsen           30-Jun-1982
{               Update Phase IV line, circuit and node parameters.
{               Add line action routine to get the physical device
{               name being used by a line (including unit number).
{               Add LOOP LINE parameters and function code.
{               Change DEST SAD from a string to a longword.
{               Change DEST NODE from a string to a longword.
{
{       V013    TMH0013         Tim Halvorsen           16-Jun-1982
```

```
{               Add SPI (Server Process) database and parameters.
{               Add DECLSERV function code.
{               Make NFB$C_LENGTH equal to NFB$L_FLDID, and not be
{               4 bytes afterwards, since it should be possible to
{               issue a ACPCONTROL function with NO fields, and without
{               any ENDOFLIST field (since it isn't required anymore).
{
{       V012    TMH0012         Tim Halvorsen           04-Apr-1982
{               Remove STRT_KEY.  Remove ERR_STRT detail code.
{               Remove NFB$C_COLLATE generic field ID.
{               Increase size of P2 context area to 64 bytes.
{               Remove obsolete function codes.
{               Rename most of the symbols and fields to make the interface
{               more understandable:
{                       NFB$V_KNO    --> NFB$V_MULT
{                       NFB$V_UPD    --> NFB$V_ERRUPD
{                       NFB$V_NOUPD --> NFB$V_NOCTX
{                       NFB$C_P2STRTLNG --> NFB$C_CTX_SIZE
{               Move $NDBDEF to NETCTL.MDL
{               Add CNT (counters) parameter for XDI, XS5 and XS9 databases.
{               Remove fields which were commented out.
{               Fix classification of CRI/PLI parameters in comment field.
{               Add key and oper fields for second search key.
{               Remove obsolete NUL parameters from all databases.
{               Add Ethernet parameters.
{               Remove obsolete FNDNEXT operator, and make FNDMIN and FNDMAX
{               unavailable through the QIO interface, since they don't work.
{               Add ERR_P5 and ERR_P6 error detail codes, so that they can
{               be returned if P5 or P6 were specified.
{               Add circuit DLM flag, to indicate X.25 DLM access
{               (corresponds to the NICE OWNER parameter).
{               Add ERR_FLAGS and ERR_OPER2 error detail codes.
{               Add symbols to make accessing field IDs and values easier
{               for BLISS programs.
{               Add DTE MAXIMUM CIRCUITS parameter.
{
{       V3-011  TMH0011         Tim Halvorsen           25-Feb-1982
{               Add X.25 parameters.
{
{       V3-010  ADE0033         A.Eldridge              01-Mar-1982
{               Increased NFB$C_DB_MAX to allow access to the X25
{               databases.
{
{       V3-009  ADE0032         A.ELdridge              15-Feb-1982
{               Added 'pipeline quota' (PIQ) to the LNI database
{
{       V3-008  ADE0031         A.Eldridge              06-Jan-1982
{               Removed the 'retransmit timer' (RTT) from the circuit
{               database.
{
{       V3-007  ADE0030         Al Eldridge             30-Nov-1981
{               Added support for proxy logins
```

```
{
{       V3-006  ??
{
{       V3-005  ADE0005         Al Eldridge             2-Sep-1981
{               Update NFBDEF to get rid of overlapping symbols
{
{       V3-004  TMH0004         Tim Halvorsen           30-Aug-1981
{               Get rid of obsolete definitions and add $NFBDEF.
{
{---
```

```
{
{  Network Function Block
{
{  This is the format of the P1 buffer used for NETACP's IO$_ACPCONTROL Qio's.
{

module $NFBDEF;
/*
/*      The following generic field identifiers are defined for all databases.
/*
constant ENDOFLIST      equals 0  prefix NFB tag $C;    /* Used to terminate the field i.d.
constant WILDCARD       equals 1  prefix NFB tag $C;    /* Field i.d. used for "match all" database searches
constant CTX_SIZE       equals 64  prefix NFB tag $C;   /* Length of context area in P2 buffer

                        /*
                        /*      The following codes are passed in the second IOSB longword to qualify
                        /*      as SS$_ILLCNTRFUNC error.
                        /*
                        /* The high order word of these error codes must be 0
                        /* so that they won't be confused with field i.d.s
constant(
      FCT               /* Unrecognized NFB$B_FCT value.
    , DB                /* Unrecognized NFB$B_DATABASE value.
    , P1                /* The P1 buffer is invalid.
    , P2                /* The P2 buffer is invalid.
    , P3                /* The P3 buffer is invalid.
    , P4                /* The P4 buffer is invalid.
    , P5                /* The P5 buffer should not have been specified.
    , P6                /* The P6 buffer should not have been specified.
    , CELL              /* Unrecognized NFB$B_CELL value.
    , OPER              /* Unrecognized NFB$B_OPER value.
    , SRCH              /* Unrecognized NFB$L_SRCH_KEY field ID
    , SRCH2             /* Unrecognized NFB$L_SRCH2_KEY field ID
    , OPER2             /* Unrecognized NFB$B_OPER2 value.
    , FLAGS             /* Undefined bits in NFB$B_FLAGS were not zero.
    ) equals 1 increment 1  prefix NFB$_E tag RR;

                        /*
                        /*      Define the P1 buffer format
                        /*

aggregate NFBDEF structure fill prefix NFB$;
    FCT byte unsigned;  /* A function code as follows:
                        /* Function codes for the NFB
                        /*  (leaving room for 20 obsolete function codes)
    constant(
          DECLNAME      /* Declare name
        , DECLOBJ       /* Declare object
        , DECLSERV      /* Declare server process available
        ) equals 21 increment 1  prefix NFB tag $C;

                        /* Resume defining function codes
```

```
                    /*  (leave room for 4 obsolete function codes)
constant(
      LOGEVENT      /* Log a network event
    , READEVENT     /* Read current raw event queue (used by EVL only)
    ) equals 28 increment 1  prefix NFB tag $C;

                    /* Resume defining function codes
                    /*  (leave room for 3 obsolete function codes)
constant(
      FC_DELETE     /* Remove an entry from the data base.
    , FC_SHOW       /* Return specified field values.
    , FC_SET        /* Set/modify the field values.
    , FC_CLEAR      /* Clear specified field values.
    , FC_ZERCOU     /* Zero (and optionally read) counters
    , FC_LOOP       /* Loop (used only to PSI to loop an X.25 line)
    ) equals 33 increment 1  prefix NFB tag $C;     /* Maximum FCT value

constant FC_MAX     equals NFB$C_FC_LOOP  prefix NFB tag $C;/* Maximum FCT value

FLAGS_OVERLAY union fill;
    FLAGS byte unsigned;            /* Miscellaneous control flags
    FLAGS_BITS structure fill;
        ERRUPD bitfield mask;       /* Update position context, even on error
        MULT bitfield mask;         /* Process as many entries as can be fit into P4
        NOCTX bitfield mask;        /* Don't update position context, even if successful
                                    /*  (used to stay on an entry for a while).  This
                                    /*  flag Overrides the ERRUPD flag.
    end FLAGS_BITS;

end FLAGS_OVERLAY;
DATABASE byte unsigned;/* A code identifying the database as follows:
                    /* ZERO is an illegal value for this field
constant(
      LNI           /* Local node
    , NDI           /* Common nodes
    , OBI           /* Network objects
    , CRI           /* Circuits
    , PLI           /* Lines
    , EFI           /* Event logging filters
    , ESI           /* Event logging sinks
    , LLI           /* Logical-links
    , XNI           /* X.25 networks
    , XGI           /* X.25 groups
    , XDI           /* X.25 DTEs
    , XS5           /* X.25 server
    , XD5           /* X.25 destinations
    , XS9           /* X.29 server
    , XD9           /* X.29 destinations
    , XTI           /* X.25 trace facility
    , XTT           /* X.25 tracepoints
    , SPI           /* Server Process
    , AJI           /* Adjacency information
```

```
    , ARI           /* Area information
                    /* (The following codes are reserved for future PSIACP
                    /*  databases.  These codes should only be used in the
                    /*  event PSIACP needs a database code before a new
                    /*  new NETACP can be supplied to support it).
    , XDTE          /* PSI reserved database
    , PSI2          /* PSI reserved database
    , PSI3          /* PSI reserved database
    , PSI4          /* PSI reserved database
    , PSI5          /* PSI reserved database
    , SDI           /* Service (DLE) information
    , XAI           /* X.25 access database
    , XXX           /* Last database definition for NFB$C_DB_MAX calc.
    ) equals 1 increment 1  prefix NFB$C_ tag DB;   /* Maximum DATABASE value

constant MAX        equals NFB$C_DB_XXX-1  prefix NFB$C_ tag DB;/* Maximum DATABASE value
OPER byte unsigned; /* Specifies the sense of the search (e.g. EQL, GEQU)
                    /* when comparing against the SRCH_KEY field.
constant(
      EQL           /* Match if SEARCH_KEY value EQL database entry field
    , GTRU          /* Match if SEARCH_KEY value GTRU database entry field
    , LSSU          /* Match if SEARCH_KEY value LSSU database entry field
    , NEQ           /* Match if SEARCH_KEY value NEQ database entry field
                    /* The following may only be used internally by NETACP
    , FNDMIN        /* Find entry with minimum key value
    , FNDMAX        /* Find entry with maximum key value
    , FNDPOS        /* Find entry position in database

    ) equals 0 increment 1  prefix NFB$C_ tag OP;/* Maximum operator function

constant MAXFCT     equals NFB$C_OP_NEQ  prefix NFB$C_ tag OP;/* Maximum operator function
constant MAXINT     equals NFB$C_OP_FNDPOS  prefix NFB$C_ tag OP;/* Maximum internal function

SRCH_KEY longword unsigned;/* Search key field identifier specifying the key used
                    /* to locate the entry in the database.  This search is
                    /* controlled by the sense of the NFB$B_OPER field.
                    /*
                    /* If this field has the value "NFB$C_WILDCARD", then
                    /* the very next entry in the list is assumed to be the
                    /* target of the search.
                    /*
                    /* If this field is not specified (zero), then it
                    /* is assumed to be NFB$C_WILDCARD (no search key).
                    /*
SRCH2_KEY longword unsigned;/* Secondary search key field ID specifying the key used
                    /* to locate the entry in the database.  This search is
                    /* controlled by the sense of the NFB$B_OPER2 field.
                    /*
                    /* If both SRCH_KEY and SRCH2_KEY are specified, then
                    /* only those database entries matching both search keys
                    /* will be processed.
                    /*
```

```
                        /* If this field is not specified (zero), then it
                        /* is assumed to be NFB$C_WILDCARD (no search key).
                        /*
    OPER2 byte unsigned;/* Specifies the sense of the search (e.g. EQL, GEQU)
                        /* when comparing against the SRCH2_KEY field.
    MBZ1 byte unsigned; /* Reserved. MBZ.
    CELL_SIZE word unsigned;/* Some of the field values found in the P4 buffer are
    constant "LENGTH" equals . prefix NFB$ tag K;/* Minimum structure size.
    constant "LENGTH" equals . prefix NFB$ tag C;/* Minimum structure size.
                        /* counted strings.  If the "cell size" is non-zero, it
                        /* indicates the number of bytes which each string in
                        /* the P4 buffer occupies.  If it is zero then strings
                        /* fields are stored as variable lengthed strings.
    FLDID longword unsigned;/* Cell containing the first field ID -- the list
                        /* of field IDs begins here and continues to the
                        /* end of the structure.
                        /*
                        /* The list may be terminated before the end of the
                        /* structure by placing the value NFB$C_ENDOFLIST
                        /* in the longword following the last field ID.
                        /*

                        /*
                        /*      Define the "field i.d." format.
                        /*
end NFBDEF;

aggregate NFBDEF1 union fill prefix NFB$;
    PARAM_ID longword unsigned;                 /* Define parameter ID longword
        PARAM_ID_BITS structure fill;
            INX bitfield mask length 16;        /* Index into semantic table
            TYP bitfield mask length 2;         /* Field type (string, bit, etc.)
            SPARE bitfield mask length 6;       /* Reserved, MBZ
            DB bitfield mask length 8;          /* Data-base i.d.
        end PARAM_ID_BITS;
        constant TYP_BIT equals 0  prefix NFB tag $C;   /* Field type for bits
        constant TYP_V equals 0  prefix NFB tag $C;     /* Field type for bits
        constant TYP_LNG equals 1  prefix NFB tag $C;   /* Field type for longwords
        constant TYP_L  equals 1  prefix NFB tag $C;    /* Field type for longwords
        constant TYP_STR equals 2  prefix NFB tag $C;   /* Field type for strings
        constant TYP_S  equals 2  prefix NFB tag $C;    /* Field type for strings

/*
/*   Define useful symbols for storing and retreiving binary and string
/*   values from the P2 and P4 buffers
/*

end NFBDEF1;

aggregate NFBDEF2 union fill prefix NFB$;
    LNG_VALUE longword unsigned;/* Longword value
end NFBDEF2;
```

```
aggregate NFBDEF3 union fill prefix NFB$;
    BIT_VALUE longword unsigned;/* Boolean value
end NFBDEF3;

aggregate NFBDEF4 union fill prefix NFB$;
    STR_COUNT word unsigned;                              /* String count field
        STR_COUNT_FIELDS structure fill;
            FILL_1 byte dimension 2 fill prefix NFBDEF tag $$;
            STR_TEXT character length 0 tag B;            /* Start of string data

/*
/*  Define identifiers for each parameter in all database
/*
/*   **  The low order 16 bits for each parameter must be unique  **
/*  ***  with respect to all other parameters in its particular   ***
/*   **  database.                                               **
```

```
/*
/* Define a field identifier index for each parameter in the NDI database.
/*
                        /*
                        /*   Boolean parameters
                        /*
        constant(
          LCK           /* Set if conditionally writable fields are not writable
        , LOO           /* Set if CNF is for a "loopback" node
        , REA           /* Set if node is reachable
        ) equals (((NFB$C_DB_NDI@24)+(NFB$C_TYP_BIT@16)+1)) increment 1  prefix NFB$C_N tag DI;

                        /*
                        /*   "Longword" Parameters
                        /*
        constant(
          TAD           /* "transformed address" - uses local node address
                        /* for the local NDI (instead of zero as does ADD)
        , CTA           /* Absolute due time for logging counters
        , ADD           /* Address
        , CTI           /* Counter timer
        , ACL           /* Active links
        , DEL           /* Delay
        , DTY           /* Destination Type
        , DCO           /* Destination Cost
        , DHO           /* Destination Hops
        , SDV           /* Service Device
        , CPU           /* CPU type
        , STY           /* Software type
        , DAD           /* Dump address
        , DCT           /* Dump count
        , OHO           /* Host
        , IHO           /* Host
        , ACC           /* Access switch (inbound, outbound, etc)
        , PRX           /* ** obsolete ** (Node proxy parameter)
        , NND           /* Next node address
        , SNV           /* Service Node Version
        , INB           /* Async Line - Inbound node type
        ) equals (((NFB$C_DB_NDI@24)+(NFB$C_TYP_LNG@16)+16)) increment 1  prefix NFB$C_N tag DI;

                        /*
                        /*   String parameters
                        /*
        constant(
          COL           /* Collating field
        , HAC           /* Node address/loop linename combination
        , CNT           /* Counters
        , NNA           /* Name
        , SLI           /* Service line
        , SPA           /* Service password
        , LOA           /* Load file
```

```
, SLO             /* Secondary loader
, TLO             /* Tertiary loader
, SID             /* Software ID
, DUM             /* Dump file
, SDU             /* Secondary dumper
, NLI             /* Loopback Line
, DLI             /* Destination Line
, PUS             /* Privileged user id
, PAC             /* Privileged account
, PPW             /* Privileged password
, NUS             /* Non-privileged user id
, NAC             /* Non-privileged account
, NPW             /* Non-privileged password
, RPA             /* Receive password
, TPA             /* Transmit password
, DFL             /* Diagnostic load file
, HWA             /* Hardware NI address (ROM address)
, LPA             /* Loop assistant NI address
, NNN             /* Next node name to destination (goes with NND)
) equals (((NFB$C_DB_NDI@24)+(NFB$C_TYP_STR@16)+64)) increment 1  prefix NFB$C_N tag DI;
```

```
/*
/* Define a field identifier index for each parameter in the LNI database.
/*
                        /*
                        /*   Boolean parameters
                        /*
        constant(
          LCK           /* Set if conditionally writable fields are not writable
        , ALI           /* Set if ALIAS INBOUND has been enabled
        ) equals (((NFB$C_DB_LNI@24)+(NFB$C_TYP_BIT@16)+1)) increment 1  prefix NFB$C_L tag NI;

                        /*
                        /*   "Longword parameters
                        /*
        constant(
          ADD           /* Address
        , ACL           /* Total number of active links
        , ITI           /* Incoming timer
        , OTI           /* Outgoing timer
        , STA           /* State
        , MLK           /* Maximum links
        , DFA           /* Delay factor
        , DWE           /* Delay weight
        , IAT           /* Inactivity timer
        , RFA           /* Retransmit factor
        , ETY           /* Executor Type
        , RTI           /* Routing timer
        , RSI           /* Routing suppression timer
        , SAD           /* Subaddress
                        /* (lower word = lower limit, upper word = upper limit)
        , MAD           /* Maximum address
        , MLN           /* Maximum lines
        , MCO           /* Maximum cost
        , MHO           /* Maximum hops
        , MVI           /* Maximum visits
        , MBU           /* Maximum buffers
        , BUS           /* Forwarding buffer size
        , LPC           /* Loop count
        , LPL           /* Loop length
        , LPD           /* Loop Data type
        , DAC           /* Default access switch (inbound, outbound, etc)
        , DPX           /* Default proxy access (inbound, outbound, etc)
        , PIQ           /* Pipeline quota
        , LPH           /* Loop help type of assistance given to loop requestors
        , BRT           /* Broadcast routing timer
        , MAR           /* Maximum areas
        , MBE           /* Maximum nonrouters on NI
        , MBR           /* Maximum routers on NI
        , AMC           /* Area maximum cost
        , AMH           /* Area maximum hops
        , SBS           /* Segment buffer size
        , ALA           /* Alias local node address (cluster address)
```

```
, ALM             /* Alias maximum links
) equals (((NFB$C_DB_LNI@24)+(NFB$C_TYP_LNG@16)+16)) increment 1  prefix NFB$C_L tag NI;

                  /*
                  /*   String parameters
                  /*
constant(
  COL             /* Collating field
, NAM             /* Local node name
, CNT             /* Counters
, IDE             /* Identification
, MVE             /* Management version
, NVE             /* Nsp version
, RVE             /* Routing version
, PHA             /* Physical NI address (current address)
) equals (((NFB$C_DB_LNI@24)+(NFB$C_TYP_STR@16)+64)) increment 1  prefix NFB$C_L tag NI;
```

```
/*
/* Define a field identifier index for each parameter in the OBI database.
/*
                        /*
                        /*   Boolean Parameters
                        /*
        constant(
          LCK           /* Set if conditionally writable fields are not writable
        , SET           /* Set if a "set" QIO has ever modified the CNF. If
                        /* not then the CNF was due to a "declare name/obect"
                        /* only and may be deleted when the declaring process
                        /* breaks the channel over which the object was declared
        , ALO           /* Alias Outgoing enabled/disabled
        , ALI           /* Alias Incoming enabled/disabled
        ) equals (((NFB$C_DB_OBIe24)+(NFB$C_TYP_BITe16)+1)) increment 1  prefix NFB$C_0 tag BI;

                        /*
                        /*   Longword Parameters
                        /*
        constant(
          LPR           /* Low order privileges
        , HPR           /* High order privileges
        , UCB           /* Owner's UCB address
        , CHN           /* Owner's channel
        , NUM           /* Number
        , PID           /* Process id
        , PRX           /* Proxy login switch (inbound, outbound, etc)
        ) equals (((NFB$C_DB_OBIe24)+(NFB$C_TYP_LNGe16)+16)) increment 1  prefix NFB$C_0 tag BI;

                        /*
                        /*   String Parameters
                        /*
        constant(
          COL           /* Collating field
        , ZNA           /* Zero obj+name identifier
        , SFI           /* Parsed file i.d.
        , IAC           /* Default inbound combined access control string
        , NAM           /* Name
        , FID           /* File id
        , USR           /* User id
        , ACC           /* Account
        , PSW           /* Password
        ) equals (((NFB$C_DB_OBIe24)+(NFB$C_TYP_STRe16)+64)) increment 1  prefix NFB$C_0 tag BI;
```

```
/*
/* Define a field identifier index for each parameter in the CRI database.
/*
/*
/*                      /* Use
/*                      /* ----
/*                      C = common
/*                      E = Executor (used by Transport)
/*                      X = Native X.25 network management
/*                      D = DECnet (not X.25)
/*
/*
/*   Boolean Parameters
/*
        constant(
          LCK           /* D Set if conditionally writable fields are
                        /*      not writable
        , SER           /* D Set if Service functions not allowed
        , BLK           /* E Blocking
        , VER_FILL      /*   Filler (VER retired)
        , DLM           /* E Circuit to be used as X.25 datalink, if set
                        /*      If clear, circuit is for X.25 native use
        ) equals (((NFB$C_DB_CRI@24)+(NFB$C_TYP_BIT@16)+1)) increment 1  prefix NFB$C_C tag RI;

                        /*
                        /*   "Longword" parameters
                        /*
        constant(
          OWPID         /* D PID of temp owner of line in service state
        , CTA           /* D Absolute due time for counter logging
        , SRV           /* D Service substate qualifier
        , STA           /* C State
        , SUB           /* C Substate
        , LCT           /* C Counter timer
        , PNA           /* E Adjacent node address
        , BLO           /* E Partner's receive block size
        , COS           /* E Cost
        , HET           /* E Hello timer
        , LIT           /* E Listen timer
        , MRC           /* E Maximum recalls
        , RCT           /* E Recall timer
        , POL           /* D Polling state
        , PLS           /* D Polling substate
        , USE           /* X Usage
        , TYP           /* C Type
        , CHN           /* X X.25 Channel
        , MBL           /* X Maximum block
        , MWI           /* X Maximum window
        , TRI           /* D Tributary
        , BBT           /* D Babble timer
        , TRT           /* D Transmit timer
        , MRB           /* D Maximum receive buffers
```

```
, MTR           /* D Maximum transmits
, ACB           /* D Active base
, ACI           /* D Active increment
, IAB           /* D Inactive base
, IAI           /* D Inactive increment
, IAT           /* D Inactive threshold
, DYB           /* D Dying base
, DYI           /* D Dying increment
, DYT           /* D Dying threshold
, DTH           /* D Dead threshold
, MST           /* D Maintenance mode state (0 => On, 1 => Off>
, XPT           /* E Transport protocol to use
, MRT           /* E Maximum routers on this NI
, RPR           /* E Router priority
, DRT           /* E Designated router on NI (node address)
, VER           /* D Verification Enabled/Disabled/Inbound on circuit
) equals (((NFB$C_DB_CRI@24)+(NFB$C_TYP_LNG@16)+16)) increment 1  prefix NFB$C_C tag RI;

                /*
                /*   String Parameters
                /*
constant(
  COL           /* D Collating field
, NAM           /* C Circuit name
, VMSNAM        /* D Device name in VMS format
, CHR           /* D Characteristics buffer for startup control QIO
, CNT           /* C Counters
, P2P           /* D Line's PhaseII partner name (for loopback)
, LOO           /* E Loopback name
, PNN           /* E Adjacent node name
, NUM           /* X Call Number
, DTE           /* X DTE
, DEVNAM        /* D Device name in VMS format, with unit included
, net           /* XD Network name
) equals (((NFB$C_DB_CRI@24)+(NFB$C_TYP_STR@16)+64)) increment 1  prefix NFB$C_C tag RI;
```

```
/*
/* Define a field identifier index for each parameter in the PLI database.
/*
/*                      C = common
/*                      L = LAPB (X.25)
/*                      D = DDCMP (not X.25)
/*                      E = Ethernet
/*
/*                      /* Use
                        /* ----
                        /*
                        /*   Boolean Parameters
                        /*
        constant(
          LCK           /* D Set if conditionally writable fields are
                        /*       not writable
        , SER           /* D Service
        , DUP           /* C Duplex (set if half)
        , CON           /* C Controller (set if loopback)
        , CLO           /* C Clock mode (set if internal)
        , SWI           /* D Async Line - Switch
        , HNG           /* D Async Line - Hangup
        ) equals (((NFB$C_DB_PLI@24)+(NFB$C_TYP_BIT@16)+1)) increment 1  prefix NFB$C_P tag LI;

                        /*
                        /*   "Longword"  Parameters
                        /*
        constant(
          CTA           /* D Absolute time for counter read and clear
        , STA           /* C State
        , SUB           /* C Substate
        , LCT           /* D Counter timer
        , PRO           /* C Protocol
        , STI           /* D Service timer
        , HTI           /* L Holdback timer
        , MBL           /* L Maximum block
        , MRT           /* L Maximum retransmits
        , MWI           /* L Maximum window
        , SLT           /* D Scheduling timer
        , DDT           /* D Dead timer
        , DLT           /* D Delay timer
        , SRT           /* D Stream timer
        , BFN           /* D Receive buffers
        , BUS           /* D Action routine returns bufsiz used for line
        , PLVEC         /* D PLVEC i.d.
        , RTT           /* D Retransmit timer
        , MOD           /* L X.25 mode (DCE, DTE, etc).
        , LPC           /* L Loop count
        , LPL           /* L Loop length
        , LPD           /* L Loop Data type
        , EPT           /* E Ethernet protocol type for datalink
        , LNS           /* D Async Line - Line speed
```

```
, BFS           /* C Line buffer size (overrides executor bufsiz)
, TPI           /* D Transmit Pipeline
) equals (((NFB$C_DB_PLI*24)+(NFB$C_TYP_LNG*16)+16)) increment 1  prefix NFB$C_P tag LI;

                /*
                /*   String Parameters
                /*
constant(
  COL           /* D Collating field
, NAM           /* C Line name
, VMSNAM        /* D Device name in VMS format
, CHR           /* D Set-mode $QIO line Characteristics buffer
, CNT           /* C Counters
, MCD           /* L Filespec for microcode dump (initiates dump)
, HWA           /* D NI hardware address (ROM address)
, DEVNAM        /* D Device name in VMS format, with unit included
, NET           /* L Network name
) equals (((NFB$C_DB_PLI*24)+(NFB$C_TYP_STR*16)+64)) increment 1  prefix NFB$C_P tag LI;
```

```
/*
/* Define a field identifier index for each parameter in the EFI database.
/*
                        /*
                        /*   Boolean Parameters
                        /*
        constant(
        LCK             /* Set if conditionally writable fields are not writable
        ) equals (((NFB$C_DB_EFI@24)+(NFB$C_TYP_BIT@16)+1)) increment 1  prefix NFB$C_E tag FI;

                        /*
                        /*   "Longword" Parameters
                        /*
        constant(
          SIN
        , SP1
        , B1
        , B2
        ) equals (((NFB$C_DB_EFI@24)+(NFB$C_TYP_LNG@16)+16)) increment 1  prefix NFB$C_E tag FI;

                        /*
                        /*   String Parameters
                        /*
        constant(
          COL           /* Collating field
        , EVE
        , SB1
        , SB2
        , SB3
        ) equals (((NFB$C_DB_EFI@24)+(NFB$C_TYP_STR@16)+64)) increment 1  prefix NFB$C_E tag FI;
```

```
/*
/* Define a field identifier index for each parameter in the ESI database.
/*
                        /*
                        /*   Boolean Parameters
                        /*
        constant(
          LCK           /* Set if conditionally writable fields are not writable
        ) equals (((NFB$C_DB_ESI*24)+(NFB$C_TYP_BIT*16)+1)) increment 1  prefix NFB$C_E tag SI;

                        /*
                        /*   "Longword" Parameters
                        /*
        constant(
          SNK
        , STA
        , SP1
        , B1
        , B2
        ) equals (((NFB$C_DB_ESI*24)+(NFB$C_TYP_LNG*16)+16)) increment 1  prefix NFB$C_E tag SI;

                        /*
                        /*   String Parameters
                        /*
        constant(
          COL           /* Collating field
        , LNA
        , SB1
        , SB2
        , SB3
        ) equals (((NFB$C_DB_ESI*24)+(NFB$C_TYP_STR*16)+64)) increment 1  prefix NFB$C_E tag SI;
```

```
/*
/* Define a field identifier index for each parameter in the LLI database.
/*
                        /*
                        /*   Boolean Parameters
                        /*
        constant(
          LCK           /* Set if conditionally writable fields are not writable
          ) equals (((NFB$C_DB_LLI@24)+(NFB$C_TYP_BIT@16)+1)) increment 1  prefix NFB$C_L tag LI;

                        /*
                        /*   Longword Parameters
                        /*
        constant(
          DLY           /* Round trip delay time
        , STA           /* State
        , LLN           /* Local link number
        , RLN           /* Remote link number
        , PNA           /* Partner's node address
        , PID           /* External Process I.D.
        , IPID          /* Internal Process I.D.
        , XWB           /* Pointer to XWB
        , CNT           /* Counters
        ) equals (((NFB$C_DB_LLI@24)+(NFB$C_TYP_LNG@16)+16)) increment 1  prefix NFB$C_L tag LI;

                        /*
                        /*   String Parameters
                        /*
        constant(
          COL           /* Collating field
        , USR           /* User name
        , PRC           /* Process name
        , PNN           /* Partner's node name
        , RID           /* Partner's process i.d.
        ) equals (((NFB$C_DB_LLI@24)+(NFB$C_TYP_STR@16)+64)) increment 1  prefix NFB$C_L tag LI;
```

```
/*
/* X.25 network parameters (part of MODULE X25-PROTOCOL)
/*
/* Define a field identifier index for each parameter in the XNI database.
/*
/*
/*   Boolean Parameters
/*
        constant(
          LCK           /* Set if conditionally writable fields are not writable
        , MNS_FILL      /* X.25 multi-network support (set if enabled) [No longer used]
        ) equals (((NFB$C_DB_XNI@24)+(NFB$C_TYP_BIT@16)+1)) increment 1  prefix NFB$C_X tag NI;

                        /*
                        /*   "Longword" Parameters
                        /*
        constant(
          CAT           /* Call timer
        , CLT           /* Clear timer
        , DBL           /* Default data
        , DWI           /* Default window
        , MBL           /* Maximum data
        , MCL           /* Maximum clears
        , MRS           /* Maximum resets
        , MST           /* Maximum restarts
        , MWI           /* Maximum window
        , RST           /* Reset timer
        , STT           /* Restart timer
        ) equals (((NFB$C_DB_XNI@24)+(NFB$C_TYP_LNG@16)+16)) increment 1  prefix NFB$C_X tag NI;

                        /*
                        /*   String Parameters
                        /*
        constant(
          COL           /* Collating field
        , netent        /* Network
        , PROF          /* Profile name
        ) equals (((NFB$C_DB_XNI@24)+(NFB$C_TYP_STR@16)+64)) increment 1  prefix NFB$C_X tag NI;
```

```
/*
/* X.25 DTE parameters (qualified by a given network)
/*
/* Define a field identifier index for each parameter in the XDI database.
/*
/*
/*   Boolean Parameters
/*
        constant(
          LCK           /* Set if conditionally writable fields are not writable
        ) equals (((NFB$C_DB_XDI@24)+(NFB$C_TYP_BIT@16)+1)) increment 1  prefix NFB$C_X tag DI;

                        /*
                        /*   "Longword" Parameters
                        /*
        constant(
          ACH           /* Active channels
        , ASW           /* Active switched
        , CTM           /* Counter timer
        , MCH           /* Maximum channels
        , STA           /* State
        , SUB           /* Substate
        , MCI           /* Maximum circuits [VMS only]
        , CAT           /* Call timer
        , CLT           /* Clear timer
        , DBL           /* Default data
        , DWI           /* Default window
        , MBL           /* Maximum data
        , MCL           /* Maximum clears
        , MRS           /* Maximum resets
        , MST           /* Maximum restarts
        , MWI           /* Maximum window
        , RST           /* Reset timer
        , STT           /* Restart timer
        , mode          /* DTE Mode
        , itt           /* Interrupt timer
        ) equals (((NFB$C_DB_XDI@24)+(NFB$C_TYP_LNG@16)+16)) increment 1  prefix NFB$C_X tag DI;

                        /*
                        /*   String Parameters
                        /*
        constant(
          COL           /* Collating field
        , DTE           /* DTE address
        , CHN           /* Channels
        , LIN           /* Line
        , dnt           /* Network
        , CNT           /* Counters
        ) equals (((NFB$C_DB_XDI@24)+(NFB$C_TYP_STR@16)+64)) increment 1  prefix NFB$C_X tag DI;
```

```
/*
/* X.25 group parameters (qualified by a given DTE)
/*
/* Define a field identifier index for each parameter in the XGI database.
/*
                        /*
                        /*   Boolean Parameters
                        /*
        constant(
          LCK           /* Set if conditionally writable fields are not writable
        ) equals (((NFB$C_DB_XGI*24)+(NFB$C_TYP_BIT*16)+1)) increment 1  prefix NFB$C_X tag GI;

                        /*
                        /*   "Longword" Parameters
                        /*
        constant(
          GNM           /* Group number
        , GTV           /* Group type
        ) equals (((NFB$C_DB_XGI*24)+(NFB$C_TYP_LNG*16)+16)) increment 1  prefix NFB$C_X tag GI;

                        /*
                        /*   String Parameters
                        /*
        constant(
          COL           /* Collating field. This field must be unique across
                        /* all entries in this database.  It consists of the
                        /* group-name string followed by the DTE address.
        , GRP           /* Group name
        , GDT           /* Group DTE address
        , gnt           /* Group Network
        ) equals (((NFB$C_DB_XGI*24)+(NFB$C_TYP_STR*16)+64)) increment 1  prefix NFB$C_X tag GI;
```

```
/*
/* X.25 server parameters (global parameters for all destinations)
/*
/* Define a field identifier index for each parameter in the XS5 database.
/*
                        /*
                        /*   Boolean Parameters
                        /*
        constant(
          LCK           /* Set if conditionally writable fields are not writable
        ) equals (((NFB$C_DB_XS5@24)+(NFB$C_TYP_BIT@16)+1)) increment 1  prefix NFB$C_X tag S5;

                        /*
                        /*   "Longword" Parameters
                        /*
        constant(
          MCI           /* Maximum circuits allowed
        , STA           /* State
        , ACI           /* Active circuits
        , CTM           /* Counter timer
        ) equals (((NFB$C_DB_XS5@24)+(NFB$C_TYP_LNG@16)+16)) increment 1  prefix NFB$C_X tag S5;

                        /*
                        /*   String Parameters
                        /*
        constant(
          COL           /* Collating field. This field must be unique across
                        /* all entries in this database.
        , CNT           /* Counters
        ) equals (((NFB$C_DB_XS5@24)+(NFB$C_TYP_STR@16)+64)) increment 1  prefix NFB$C_X tag S5;
```

```
/*
/* X.25 destination parameters (part of MODULE X25-SERVER)
/*
/* Define a field identifier index for each parameter in the XD5 database.
/*
                        /*
                        /*   Boolean Parameters
                        /*
        constant(
          LCK           /* Set if conditionally writable fields are not writable
        ) equals (((NFB$C_DB_XD5e24)+(NFB$C_TYP_BITe16)+1)) increment 1  prefix NFB$C_X tag D5;

                        /*
                        /*   "Longword" Parameters
                        /*
        constant(
          PRI           /* Priority
        , SAD           /* Subaddress range
                        /* (lower word = lower limit, upper word = upper limit)
        , NOD           /* Remote node address containing server (gateways only)
        , red           /* Redirect reason
        ) equals (((NFB$C_DB_XD5e24)+(NFB$C_TYP_LNGe16)+16)) increment 1  prefix NFB$C_X tag D5;

                        /*
                        /*   String Parameters
                        /*
        constant(
          COL           /* Collating field. This field must be unique across
                        /* all entries in this database.
        , DST           /* Destination DTE address
        , CMK           /* Call mask
        , CVL           /* Call value
        , GRP           /* Group name
        , SDTE          /* Sending DTE address (formally number)
        , OBJ           /* && Object name
        , FIL           /* Command procedure to execute when starting object
        , USR           /* User name
        , PSW           /* Password
        , ACC           /* Account
        , cdte          /* Called DTE
        , rdte          /* Receiving DTE
        , net           /* Network
        , emk           /* Extension mask
        , evl           /* Extension value
        , acl           /* ACL, a list of ACE'structure, parto of ORB
        , idte          /* Incoming address
        ) equals (((NFB$C_DB_XD5e24)+(NFB$C_TYP_STRe16)+64)) increment 1  prefix NFB$C_X tag D5;
```

```
/*
/* X.29 server parameters (global parameters for all destinations)
/*
/* Define a field identifier index for each parameter in the XS9 database.
/*
                        /*
                        /*   Boolean Parameters
                        /*
        constant(
          LCK           /* Set if conditionally writable fields are not writable
        ) equals (((NFB$C_DB_XS9@24)+(NFB$C_TYP_BIT@16)+1)) increment 1  prefix NFB$C_X tag S9;

                        /*
                        /*   "Longword" Parameters
                        /*
        constant(
          MCI           /* Maximum circuits allowed
        , STA           /* State
        , ACI           /* Active circuits
        , CTM           /* Counter timer
        ) equals (((NFB$C_DB_XS9@24)+(NFB$C_TYP_LNG@16)+16)) increment 1  prefix NFB$C_X tag S9;

                        /*
                        /*   String Parameters
                        /*
        constant(
          COL           /* Collating field. This field must be unique across
                        /* all entries in this database.
        , CNT           /* Counters
        ) equals (((NFB$C_DB_XS9@24)+(NFB$C_TYP_STR@16)+64)) increment 1  prefix NFB$C_X tag S9;
```

```
/*
/* X.29 destination parameters (part of MODULE X29-SERVER)
/*
/* Define a field identifier index for each parameter in the XD9 database.
/*
                        /*
                        /*   Boolean Parameters
                        /*
        constant(
          LCK           /* Set if conditionally writable fields are not writable
        ) equals (((NFB$C_DB_XD9@24)+(NFB$C_TYP_BIT@16)+1)) increment 1  prefix NFB$C_X tag D9;

                        /*
                        /*   "Longword" Parameters
                        /*
        constant(
          PRI           /* Priority
        , SAD           /* Subaddress range
                        /* (lower word = lower limit, upper word = upper limit)
        , NOD           /* Remote node address containing server (gateways only)
        , red           /* Redirect reason
        ) equals (((NFB$C_DB_XD9@24)+(NFB$C_TYP_LNG@16)+16)) increment 1  prefix NFB$C_X tag D9;

                        /*
                        /*   String Parameters
                        /*
        constant(
          COL           /* Collating field. This field must be unique across
                        /* all entries in this database.
        , DST           /* Destination DTE address
        , CMK           /* Call mask
        , CVL           /* Call value
        , GRP           /* Group name
        , sdte          /* Sending DTE
        , OBJ           /* && Object name
        , FIL           /* Command procedure to execute when starting object
        , USR           /* User name
        , PSW           /* Password
        , ACC           /* Account
        , cdte          /* Caller DTE
        , rdte          /* Receiving DTE
        , net           /* Network
        , emk           /* Extension mask
        , evl           /* Extension value
        , acl           /* ACL, a list of ACE'structure, parto of ORB
        , idte          /* Incoming address
        ) equals (((NFB$C_DB_XD9@24)+(NFB$C_TYP_STR@16)+64)) increment 1  prefix NFB$C_X tag D9;
```

```
/*
/* X.25 tracing facility (global) parameters.
/*
/* Define a field identifier index for each parameter in the XTI database.
/*
                        /*
                        /*   Boolean Parameters
                        /*
        constant(
          LCK           /* Set if conditionally writable fields are not writable
        ) equals (((NFB$C_DB_XTI@24)+(NFB$C_TYP_BIT@16)+1)) increment 1  prefix NFB$C_X tag TI;

                        /*
                        /*   "Longword" Parameters
                        /*
        constant(
          STA           /* State
        , BFZ           /* Buffer size
        , CPL           /* Capture limit
        , MBK           /* Maximum blocks/file
        , MBF           /* Maximum number of buffers
        , MVR           /* Maximum trace file version number
        ) equals (((NFB$C_DB_XTI@24)+(NFB$C_TYP_LNG@16)+16)) increment 1  prefix NFB$C_X tag TI;

                        /*
                        /*   String Parameters
                        /*
        constant(
          COL           /* Collating field. This field must be unique across
                        /* all entries in this database.
        , FNM           /* Trace file name
        ) equals (((NFB$C_DB_XTI@24)+(NFB$C_TYP_STR@16)+64)) increment 1  prefix NFB$C_X tag TI;
```

```
/*
/* X.25 tracpoint (local) parameters.
/*
/* Define a field identifier index for each parameter in the XTT database.
/*
                        /*
                        /*    Boolean Parameters
                        /*
        constant(
          LCK           /* Set if conditionally writable fields are not writable
        ) equals (((NFB$C_DB_XTT*24)+(NFB$C_TYP_BIT*16)+1)) increment 1  prefix NFB$C_X tag TT;

                        /*
                        /*    "Longword" Parameters
                        /*
        constant(
          TST           /* State
        , CPS           /* Capture size
        ) equals (((NFB$C_DB_XTT*24)+(NFB$C_TYP_LNG*16)+16)) increment 1  prefix NFB$C_X tag TT;

                        /*
                        /*    String Parameters
                        /*
        constant(
          COL           /* Collating field. This field must be unique across
                        /* all entries in this database.
        , TPT           /* Tracepoint name
        ) equals (((NFB$C_DB_XTT*24)+(NFB$C_TYP_STR*16)+64)) increment 1  prefix NFB$C_X tag TT;
```

```
/*
/* X.25 Access (qualified by a given network)
/*
/* Define a field identifier index for each parameter in the XAI database.
/*
                        /*
                        /*   Boolean Parameters
                        /*
        constant(
          LCK           /* Set if conditionally writable fields are not writable
        ) equals (((NFB$C_DB_XAI*24)+(NFB$C_TYP_BIT*16)+1)) increment 1  prefix NFB$C_X tag AI;

                        /*
                        /*   "Longword" Parameters
                        /*
        constant(
          NDA           /* Node address
        ) equals (((NFB$C_DB_XAI*24)+(NFB$C_TYP_LNG*16)+16)) increment 1  prefix NFB$C_X tag AI;

                        /*
                        /*   String Parameters
                        /*
        constant(
          COL           /* Collating field
        , NET           /* Network
        , USR           /* User id
        , PSW           /* Password
        , ACC           /* Account
        , NOD           /* Node id
        ) equals (((NFB$C_DB_XAI*24)+(NFB$C_TYP_STR*16)+64)) increment 1  prefix NFB$C_X tag AI;
```

```
/*
/* X.25 Security (qualified by a given network)
/*
/* Define a field identifier index for each parameter in the XDTE database.
/*
                        /*
                        /*   Boolean Parameters
                        /*
        constant (
          LCK           /* Set if conditionally writable fields are not writable
        ) equals (((NFB$C_DB_XDTE@24)+(NFB$C_TYP_BIT@16)+1)) increment 1 prefix NFB$C_ tag XDTE;

                        /*    C(,$C_XDTE_,(((NFB$C_DB_XDTE@24)+(NFB$C_TYP_LNG@16)+16)),1
                        /*
                        /*   "Longword" Parameters
                        /*
                        /*    )

                        /*
                        /*   String Parameters
                        /*
        constant (
          COL           /* Collating field
        , NET           /* Network
        , DTE           /* DTE address
        , ID            /* ID list, ARB rights list
        , ACL           /* ACL, a list of ACE's, part of ORB
        ) equals (((NFB$C_DB_XDTE@24)+(NFB$C_TYP_STR@16)+64)) increment 1 prefix NFB$C_ tag XDTE;
```

```
/*
/* Define SPI (Server Process) parameters
/*
                        /*
                        /*   Boolean Parameters
                        /*
        constant(
          LCK           /* Set if conditionally writable fields are not writable
        , PRL           /* Proxy flag which initially started server process
        ) equals (((NFB$C_DB_SPI@24)+(NFB$C_TYP_BIT@16)+1))
        increment 1  prefix NFB$C_S tag PI;

                        /*
                        /*   Longword Parameters
                        /*
        constant(
          PID           /* Server PID
        , IRP           /* IRP of waiting DECLSERV QIO (0 if process active)
        , CHN           /* Channel associated with DECLSERV IRP
        , RNA           /* Remote node address which initially started server
        ) equals (((NFB$C_DB_SPI@24)+(NFB$C_TYP_LNG@16)+16)) increment 1  prefix NFB$C_S tag PI;

                        /*
                        /*   String Parameters
                        /*
        constant(
          COL           /* Collating field
        , ACS           /* ACS used to initally start server process
        , RID           /* Remote user ID which initially started server
        , SFI           /* Last (current) SFI given to server process
        , NCB           /* Last (current) NCB given to server process
        , PNM           /* Last (current) process name given to server
        ) equals (((NFB$C_DB_SPI@24)+(NFB$C_TYP_STR@16)+64)) increment 1  prefix NFB$C_S tag PI;
```

```
/*
/* Define AJI (Adjacency) parameters
/*
                        /*
                        /*   Boolean Parameters
                        /*
        constant(
          LCK           /* Set if conditionally writable fields are not writable
        , REA           /* Reachable (set if two-way communication established)
        ) equals (((NFB$C_DB_AJI@24)+(NFB$C_TYP_BIT@16)+1)) increment 1  prefix NFB$C_A tag JI;

                        /*
                        /*   Longword Parameters
                        /*
        constant(
          ADD           /* Node address
        , TYP           /* Node type
        , LIT           /* Listen timer for this adjacency
        , BLO           /* Partner's block size
        , RPR           /* Partner's router priority (on NI)
        ) equals (((NFB$C_DB_AJI@24)+(NFB$C_TYP_LNG@16)+16)) increment 1  prefix NFB$C_A tag JI;

                        /*
                        /*   String Parameters
                        /*
        constant(
          COL           /* Collating field
        , NNA           /* Node name
        , CIR           /* Circuit name
        ) equals (((NFB$C_DB_AJI@24)+(NFB$C_TYP_STR@16)+64)) increment 1  prefix NFB$C_A tag JI;
```

```
/*
/* Define SDI (Service DLE) parameters
/*
                        /*
                        /*   Boolean Parameters
                        /*
        constant(
          LCK           /* Set if conditionally writable fields are not writable
        ) equals (((NFB$C_DB_SDI*24)+(NFB$C_TYP_BIT*16)+1)) increment 1  prefix NFB$C_S tag DI;

                        /*
                        /*   Longword Parameters
                        /*
        constant(
          SUB           /* Service substate
        , PID           /* PID of process owning this DLE link
        ) equals (((NFB$C_DB_SDI*24)+(NFB$C_TYP_LNG*16)+16)) increment 1  prefix NFB$C_S tag DI;

                        /*
                        /*   String Parameters
                        /*
        constant(
          COL           /* Collating field
        , CIR           /* Circuit name
        , PHA           /* Service physical address (BC only)
        , PRC           /* Name of process owning this DLE link
        ) equals (((NFB$C_DB_SDI*24)+(NFB$C_TYP_STR*16)+64)) increment 1  prefix NFB$C_S tag DI;
```

```
/*
/* Define the AREA database (read only) for level 2 Phase IV routers only.
/*
                        /*
                        /*   Boolean parameters
                        /*
        constant(
          LCK           /* Set if conditionally writable fields are not writable
        , REA           /* Set if node is reachable
        ) equals (((NFB$C_DB_ARI@24)+(NFB$C_TYP_BIT@16)+1)) increment 1  prefix NFB$C_A tag RI;

                        /*
                        /*   "Longword" Parameters
                        /*
        constant(
          ADD           /* Address
        , DCO           /* Destination Cost
        , DHO           /* Destination Hops
        , NND           /* Next node address
        ) equals (((NFB$C_DB_ARI@24)+(NFB$C_TYP_LNG@16)+16)) increment 1  prefix NFB$C_A tag RI;

                        /*
                        /*   String parameters
                        /*
        constant(
          COL           /* Collating field
        , DLI           /* Circuit used for normal traffic to area
        ) equals (((NFB$C_DB_ARI@24)+(NFB$C_TYP_STR@16)+64)) increment 1  prefix NFB$C_A tag RI;

        end STR_COUNT_FIELDS;
end NFBDEF4;

end_module $NFBDEF;
```

```
module $DRDEF;
/*
/* DISCONNECT REASONS
/*
constant DR_NORMAL      equals 0  prefix NET tag $C;     /* NO ERROR (SYNCH DISCONNECT)
constant DR_RSU equals 1  prefix NET tag $C;             /* COULDN'T ALLOCATE UCB ADDRESS
constant DR_NONODE      equals 2  prefix NET tag $C;     /* Unrecognized node name
constant DR_SHUT        equals 3  prefix NET tag $C;     /* NODE OR LINE SHUTTING DOWN
constant DR_NOBJ        equals 4  prefix NET tag $C;     /* UNKNOWN OBJECT TYPE OR PROCESS
constant DR_FMT equals 5  prefix NET tag $C;             /* ILLEGAL PROCESS NAME FIELD
constant DR_BUSY        equals 6  prefix NET tag $C;     /* Object too busy
constant DR_PROTCL      equals 7  prefix NET tag $C;     /* GENERAL PROTOCOL ERROR
constant DR_THIRD       equals 8  prefix NET tag $C;     /* THIRD PARTY DISCONNECT
constant DR_ABORT       equals 9  prefix NET tag $C;     /* DISCONNECT ABORT
constant DR_IVNODE      equals 2  prefix NET tag $C;     /* Invalid node name format
constant DR_NONZ        equals 21  prefix NET tag $C;    /* NON-ZERO DST ADDRESS
constant DR_BADLNK      equals 22  prefix NET tag $C;    /* INCONSISTENT DSTLNK
constant DR_ZERO        equals 23  prefix NET tag $C;    /* ZERO SOURCE ADDRESS
constant DR_BADFC       equals 24  prefix NET tag $C;    /* FCVAL ILLEGAL
constant DR_NOCON       equals 32  prefix NET tag $C;    /* NO CONNECT SLOTS AVAILABLE
constant DR_ACCESS      equals 34  prefix NET tag $C;    /* INVALID ACCESS CONTROL
constant DR_BADSRV      equals 35  prefix NET tag $C;    /* LOGICAL LINK SERVICES MISMATCH
constant DR_ACCNT       equals 36  prefix NET tag $C;    /* INVALID ACCOUNT INFORMATION
constant DR_SEGSIZ      equals 37  prefix NET tag $C;    /* SEGSIZE TOO SMALL
constant DR_EXIT        equals 38  prefix NET tag $C;    /* USER EXIT OR TIMEOUT
constant DR_NOPATH      equals 39  prefix NET tag $C;    /* NO PATH TO DESTINATION NODE
constant DR_LOSS        equals 40  prefix NET tag $C;    /* LOSS OF DATA HAS OCCURRED
constant DR_NOLINK      equals 41  prefix NET tag $C;    /* ILLEGAL MSG FOR LINK NOLINK STATE
constant DR_CONF        equals 42  prefix NET tag $C;    /* REAL DISCONNECT CONFIRM
constant DR_IMLONG      equals 43  prefix NET tag $C;    /* IMAGE DATA FIELD TOO LONG

end_module $DRDEF;
```

5 NSPMSGDEF SDL FILES

```
                MODULE  $nspmsgdef;
(*
(*              NSPMSGDEF.SDL   - NSP and Transport Message Definitions
(*              Version 'V04-001'
(*
(*****************************************************************************
(*                                                                           *
(*  COPYRIGHT (c) 1978, 1980, 1982, 1984, 1986  BY                           *
(*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                   *
(*  ALL RIGHTS RESERVED.                                                     *
(*                                                                           *
(*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED    *
(*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE    *
(*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER    *
(*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY    *
(*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY    *
(*  TRANSFERRED.                                                             *
(*                                                                           *
(*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE    *
(*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT    *
(*  CORPORATION.                                                             *
(*                                                                           *
(*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS    *
(*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                  *
(*                                                                           *
(*                                                                           *
(*****************************************************************************
(*
(*
(*  AUTHOR:             Alan D. Eldridge        1-April-1982
(*
(*  MODIFIED BY:
(*
(*      V04-001 TRC0004         Terry Cassidy   15-Nov-1984
(*              Pass all relevant data from long data packet format messages
(*              to event logger.
(*      V03-005 RNG0004         Rod N. Gamache  19-Nov-1982
(*              Change ENDNODE format messages to LONG format
(*              messages. Change RTS bit definitions in LONG format
(*              messages to be in the flags byte, rather than the
(*              service class.
(*
(*      V03-004 RNG0003         Rod N. Gamache  11-Oct-1982
(*              Add Phase IV Level 2 router definitions
(*
(*      V03-003 RNG0002         Rod N. Gamache  29-Sep-1982
(*              Add Phase IV endnode definitions
(*
(*      V03-001 RNG0001         Rod N. Gamache  14-Jul-1982
(*              Add Phase IV transport definitions
(*
```

```
{++
{
{  A thumbnail sketch of the NSP message formats is as follows:
{
{
{  <0eb0 0000><4b_LINK>(2b_ACK)<2b_SEG><DATA>                          DATA MSG
{  <0011 0000><4b_LINK>(2b_ACK)<2b_SEG><u16_DATA>                      INT. MSG
{  <0001 0000><4b_LINK>(2b_ACK)<2b_SEG><2b_FLOW>                       L.S. MSG
{
{  <0000 0100><4b_LINK><2b_ACK>                                        DATA ACK
{  <0001 0100><4b_LINK><2b_ACK>                                        OTH. ACK
{  <0010 0100><2b_DST>                                                 CA
{
{  <0001 1000><2k_0><2b_SRC><1b_SRV><1b_INFO><2b_SEGSIZ><CTL>          CI
{  <0010 1000><4b_LINK><1b_SRV><1b_INFO><2b_SEGSIZ><i16_DATA>          CC
{  <0011 1000><4b_LINK><2b_REA><i16_DATA>                              DI
{  <0100 1000><4b_LINK><2b_REA>                                        DC
{  <0100 1000><2b_DST><2k_0><2k_1>                                     CT
{  <0100 1000><4b_LINK><2k_42>                                         DT
{  <0100 1000><4b_LINK><2k_41>                                         NLT
{
{  <0101 1000>------                                                   START
{
{  <4b_LINK    ::=  <2b_DST><2b_SRC>              link address, not = 0
{  <2b_ACK     ::=  <1001><12 bit seg number>     if NAK
{                   <1000><12 bit seg number>     if ACK
{  <2b_SEG>    ::=  <0000><12 bit seg number>
{  <2b_FLOW>   ::=  <0000><2 bit subchannel><2 bit mode><1 byte count>
{                         0 => data          00 => no change
{                         1 => interrupt     01 => stop
{                                            10 => start
{
{  <1b_SRV>    ::=  <00000001>                    if no flow control
{                   <00000101>                    if segment flow control
{                   <00001001>                    if message flow control
{  <1b_INFO>   ::=  <00000001>                    if NSP V3.1
{                   <00000000>                    if NSP V3.2
{
{
{  <CTL>       ::=  <DNAME><SNAME><00vv00da><ACCOUNT><i16_DATA>
{                                             if a      if d
{                                  vv = Session control version
{                                   0 = Version 1.0
{                                       All other are reserved
{                    <DNAME> ::= <NAME>
{                    <SNAME> ::= <NAME>
{                    <NAME>  ::= <1k_0><1b_objtyp>  objtyp not= 0
{                                <1k_1><1k_0><i16_desc>
{                                <1k_2><1k_0><2b_gcod><2b_ucod><i12_desc>
{                    <ACCT>  ::= <i39_id><i39_psw><i39_acc>
{
```

```
{
{--
AGGREGATE  nspmsg  STRUCTURE  PREFIX nsp$ TAG $$;
    {
    {
    {  Define message codes -- first byte of message
    {
    {
    CONSTANT  msg_ci     EQUALS %x18 TAG c;      { Connect Initiate
    CONSTANT  msg_ca     EQUALS %x24 TAG c;      { Connect Acknowledge
    CONSTANT  msg_cc     EQUALS %x28 TAG c;      { Connect Confirm
    CONSTANT  msg_data   EQUALS %x00 TAG c;      { Data
    CONSTANT  msg_int    EQUALS %x30 TAG c;      { Interrupt
    CONSTANT  msg_ls     EQUALS %x10 TAG c;      { Link Service
    CONSTANT  msg_liack  EQUALS %x14 TAG c;      { LS/INT Ack
    CONSTANT  msg_dtack  EQUALS %x04 TAG c;      { Data ACK
    CONSTANT  msg_di     EQUALS %x38 TAG c;      { Disconnect Initiate
    CONSTANT  msg_dc     EQUALS %x48 TAG c;      { Disconnect Confirm
                                                 {
    {
    {
    {  Define maximum message header sizes
    {
    {
    CONSTANT  hsz_ci     EQUALS  240 TAG c;      {&Connect Initiate
    CONSTANT  hsz_ca     EQUALS    3 TAG c;      { Connect Acknowledge
    CONSTANT  hsz_cc     EQUALS  100 TAG c;      {&Connect Confirm
    CONSTANT  hsz_data   EQUALS    9 TAG c;      { Data
    CONSTANT  hsz_int    EQUALS    9 TAG c;      { Interrupt
    CONSTANT  hsz_ls     EQUALS    9 TAG c;      { Link Service
    CONSTANT  hsz_ack    EQUALS    7 TAG c;      { LS/INT or Data Ack
    CONSTANT  hsz_di     EQUALS   22 TAG c;      { Disconnect Initiate
    CONSTANT  hsz_dc     EQUALS   22 TAG c;      { Disconnect Confirm
    CONSTANT  hsz_cd     EQUALS  240 TAG c;      {&Maximum of all connect
                                                 {  or disconnect messages
                                                 {
    qual        UNION;                           { Miscellaneous message fields
                                                 {
      qual_msg      STRUCTURE TAG $$;            { Misc. MSG field qualifiers
        msg_sp1     BITFIELD  LENGTH 4;          { Skip constant part of field
        msg_li      BITFIELD  MASK;              { Set if not DATA subchannel msg
        msg_int     BITFIELD  MASK;              { Set if INT rather than LS msg
        dstlnk      WORD;                        { Destination link address
        srclnk      WORD;                        { Source link address
        END qual_msg;                            {
                                                 {
      qual_data     STRUCTURE TAG $$;            { Qualifiers to NSP$C_MSG_DATA
        data_sp     BITFIELD  LENGTH 5;          { Skip constant part of field
        data_bom    BITFIELD  MASK;              { Set if first seg in message
        data_eom    BITFIELD  MASK;              { Set if last seg in message
        data_ovfw   BITFIELD  MASK;              { Set if no further segs could
        END qual_data;                           {  be buffered without overflow
```

```
                                                    {  of the session ctl buffer -
                                                    {  INTERNAL ONLY, not in NSP
                                                    {
qual_ack      STRUCTURE TAG $$;                     { Qualifiers to all ACK fields
  ack_num     BITFIELD  MASK   LENGTH 12;           { Seg number being ACK'd
  ack_nak     BITFIELD  MASK;                       { Set if this is a NAK
  ack_sp2     BITFIELD  LENGTH 2;                   { Skip reserved field
  ack_valid   BITFIELD  MASK;                       { Set to identify field as an
  END qual_ack;                                     {  ACK and not just a segment #
                                                    {
qual_srv      STRUCTURE TAG $$;                     { Qualifiers to SERVICES field
                                                    { in CI and CC messages
  srv_01      BITFIELD  MASK   LENGTH 2;            { Must be value "01"
  srv_flw     BITFIELD  MASK   LENGTH 2;            { Receiver's flow control mode
  CONSTANT    srv_nfc   EQUALS 0 TAG c;             {  no flow control
  CONSTANT    srv_sfc   EQUALS 1 TAG c;             {  segment flow control
  CONSTANT    srv_mfc   EQUALS 2 TAG c;             {  message flow control
  srv_sp1     BITFIELD  MASK   LENGTH 3;            { Reserved bits
  srv_ext     BITFIELD  MASK;                       { Set if field extends to next
                                                    { byte
  CONSTANT    srv_req   EQUALS %B11110011           { Mask of SERVICE bits which
                        TAG m;                      { must have a know value
  CONSTANT    srv_req   EQUALS %B00000001           { ...that known value
                        TAG c;
  END qual_srv;                                     {
                                                    {
qual_inf      STRUCTURE TAG $$;                     { Qualifiers to INFO field in
                                                    { CI and CC messages
  inf_ver     BITFIELD  MASK   LENGTH 2;            { NSP version number
  CONSTANT    inf_v32   EQUALS 0 TAG c;             {  Version 3.2
  CONSTANT    inf_v31   EQUALS 1 TAG c;             {  Version 3.1
  CONSTANT    inf_v33   EQUALS 2 TAG c;             {  Version 3.3
                                                    {  "3" is reserved
  END qual_inf;                                     {
                                                    {
qual_flw      STRUCTURE TAG $$;                     { Link Service message LSFLAGS
                                                    { field definition
  flw_mode    BITFIELD  MASK   LENGTH 2;            { Back-pressure value field
  CONSTANT    flw_nop   EQUALS 0 TAG c;             {  no-change
  CONSTANT    flw_xoff  EQUALS 1 TAG c;             {  stop flow
  CONSTANT    flw_xon   EQUALS 2 TAG c;             {  start flow
                                                    {  "3" is reserved
  flw_chan    BITFIELD  MASK   LENGTH 2;            { Sub channel selector
  CONSTANT    flw_data  EQUALS 0 TAG c;             {  Data subchannel
  CONSTANT    flw_int   EQUALS 1 TAG c;             {  Interrupt subchannel
                                                    {  "2" and "3" are reserved
  flw_drv     BITFIELD  MASK   LENGTH 4;            { Define the remainder of the
                                                    { field for Netdriver internal
                                                    { flags
  END qual_flw;                                     {
                                                    {
qual_altflw   STRUCTURE TAG $$;                     { Alternate Link Service
```

```
                                        ( message LSFLAGS
    (
    (  These flags define the LSFLAGS in an alternate way to "qual_flw"
    (  above.  It depicts the same information, but takes advantage of
    (  the fact that not all values of the 2-bit fields are used and hence
    (  1-bit flags can be used rather than constants.
    (
    (  The first 4 bits are part of NSP.  The last 4 bits are Netdriver
    (  internal flags.
    (
    flw_xoff    BITFIELD  MASK   LENGTH 1;   ( Stop flow on DATA subchannel
    flw_xon     BITFIELD  MASK   LENGTH 1;   ( Start "    "  "        "
    flw_lisub   BITFIELD  MASK   LENGTH 1;   ( Flow control count is for the
                                             ( Link Service/Interrupt, and
                                             ( not the DATA, subchannel
    flw_sp1     BITFIELD  MASK   LENGTH 1;   ( Spare
    flw_inuse   BITFIELD  MASK   LENGTH 1;   ( XWB$B_X_FLW is in use
    flw_int     BITFIELD  MASK   LENGTH 1;   ( XWB$B_X_FLW describes an
                                             ( "Interrupt", not a "Link-
                                             (  Service", message
    flw_sp2     BITFIELD  MASK   LENGTH 1;   ( Spare
    flw_sp3     BITFIELD  MASK   LENGTH 1;   ( Spare
    END qual_altflw;                         (

  END qual;

END nspmsg ;
```

```
{
{
{  Phase III Routing Message Definitions
{
{
{  The following is a thumbnail sketch of the possibilities for the first byte
{  in a received message:
{
{
{       <0000 1000>     Phase II NOP
{       <0101 1000>     Phase II Start
{
{       <0100 xx10>     Phase II route header
{       <000x x010>     Phase III route header
{
{       <000x x010>     Phase IV non-broadcast circuit route header
{       <00xx 0x10>     Phase IV broadcast circuit route header
{
{       <0000 0001>     Phase III init
{       <0000 0011>     Phase III verification
{       <0000 0101>     Phase III hello message
{       <0000 0111>     Phase III routing message
{
{       <0000 1001>     Phase IV Level 2 routing message
{       <0000 1011>     Phase IV broadcast circuit Router hello message
{       <0000 1101>     Phase IV broadcast circuit Endnode hello message
{
{
{
AGGREGATE  tr3msg  STRUCTURE  PREFIX tr3$ TAG $$;
    {
    {
    {  Define message codes --  first byte of message  (DECnet calls these
    {                           "control flags")
    {
    {
    CONSTANT  msg_init  EQUALS %x01 TAG c;      { "Initialization" message
    CONSTANT  msg_verf  EQUALS %x03 TAG c;      { "Verification" message
    CONSTANT  msg_hello EQUALS %x05 TAG c;      { "Hello" message
    CONSTANT  msg_rout  EQUALS %x07 TAG c;      { "Routing" message
    CONSTANT  msg_data  EQUALS %x02 TAG c;      { Normal route-thru message
                                                {   (without qualifiers)
    CONSTANT  msg_str2  EQUALS %x58 TAG c;      { Phase II "Start" message
    CONSTANT  msg_nop2  EQUALS %x08 TAG c;      { Phase II "Nop" message

    {
    {
    {  Define message header sizes where applicable
    {
    {
    CONSTANT  hsz_data  EQUALS   6  TAG c;      { Normal route-thru message
```

```
{
{
{  Define qualifiers to the various message codes
{
{
qual        UNION;                          { Miscellaneous message fields

  qual_msg     STRUCTURE TAG $$;             { Common qualifiers
    msg_ctl    BITFIELD  MASK LENGTH 1;      { Set on Phase III control msgs
                                             { - clear on all other messages
    msg_rth    BITFIELD  MASK LENGTH 1;      { Set if a Phase II or III
                                             { route-header,  clear if a
                                             { Phase II control message
    end qual_msg;                            {

  qual_rtflg   STRUCTURE TAG $$;             { Route-header qualifiers
    rtflg_012  BITFIELD       LENGTH 3;      { Must have the value 010
    rtflg_rqr  BITFIELD  MASK LENGTH 1;      { Set if "return-to-sender" on
                                             { error is requested
    rtflg_rts  BITFIELD  MASK LENGTH 1;      { Set if message is being
                                             { "returned-to-sender"
    rtflg_5    BITFIELD       LENGTH 1;      { Must be clear
    rtflg_ph2  BITFIELD  MASK LENGTH 1;      { Set if Phase II route-header
    rtflg_7    BITFIELD       LENGTH 1;      { Must be clear
    END qual_rtflg;

  END qual;

END tr3msg;
```

```
{
{
{  Phase IV Routing Message Definitions
{
{
{  The following is a thumbnail sketch of the possibilities for the first byte
{  in a received message:
{
{
{       <000x x010>     Phase IV non-broadcast circuit route header
{       <00xx 0x10>     Phase IV broadcast circuit route header
{
{       <0000 1001>     Phase IV Level 2 routing message
{       <0000 1011>     Phase IV broadcast circuit Router hello message
{       <0000 1101>     Phase IV broadcast circuit Endnode hello message
{
{
{
AGGREGATE  tr4msg  STRUCTURE  PREFIX tr4$ TAG $$;
    #id_length = 6;                                 ( Broadcast ID length

    flags STRUCTURE;                                ( Flags
    {
    {
    {  Define qualifiers to message codes
    {
    {
        rtflg_01  BITFIELD       LENGTH 2;          ( Must have the value 10
        rtflg_lng BITFIELD  MASK LENGTH 1;          ( Set if Long format message
        rtflg_rqr BITFIELD  MASK LENGTH 1;          ( Set if return-to-sender requested
        rtflg_rts BITFIELD  MASK LENGTH 1;          ( Set if message is being
                                                    ( "returned-to-sender"
        rtflg_ini BITFIELD  MASK LENGTH 1;          ( Set if Intra-NI message
                                                    (  (on route-thru messages)
        rtflg_ver BITFIELD       LENGTH 2;          ( Route header version number
        END flags;                                  (
    {
    {
    {  Define message codes --  first byte of message  (DECnet calls these
    {                           "control flags")
    {
    {
    CONSTANT  msg_bcrhel EQUALS %xB TAG c;          ( Broadcast Circuit Router
                                                    (  "Hello" message
    CONSTANT  msg_bcehel EQUALS %xD TAG c;          ( Broadcast Circuit Endnode
                                                    (  "Hello" message
                                                    (  (without qualifiers)
    CONSTANT  msg_rdata EQUALS %x02 TAG c;          ( Normal route-thru message
    CONSTANT  msg_ldata EQUALS %x06 TAG c;          ( Long header data message

    d_area    BYTE UNSIGNED;                        ( Reserved
```

```
    d_subarea BYTE UNSIGNED;                        { Reserved
    d_id      STRUCTURE DIMENSION (#id_length);     { Destination ID
        filler     BYTE UNSIGNED FILL;              {
        END d_id;                                   {
    s_area    BYTE UNSIGNED;                        { Reserved
    s_subarea BYTE UNSIGNED;                        { Reserved
    s_id      STRUCTURE DIMENSION (#id_length);     { Source ID
        filler     BYTE UNSIGNED FILL;              {
        END s_id;                                   {
    nl2       BYTE UNSIGNED;                        { Next level 2 router, reserved
    visit_ct  BYTE UNSIGNED;                        { Visit Count
    s_class   STRUCTURE;                            { Service Class, reserved
        sclass_metr  BITFIELD       LENGTH 1;       { Metric - RESERVED
        sclass_1     BITFIELD       LENGTH 1;       { Must be clear - RESERVED
        sclass_ls    BITFIELD       LENGTH 1;       { Load splitting - RESERVED
        sclass_suba  BITFIELD       LENGTH 1;       { Sub Area - RESERVED
        sclass_bc    BITFIELD       LENGTH 1;       { Broadcast - RESERVED
        sclass_57    BITFIELD       LENGTH 3;       { Must be clear - RESERVED
        END s_class;                                {
    pt        BYTE UNSIGNED;                        { Protocol Type, reserved

    {
    {
    {  Define constants
    {
    {
    CONSTANT  T3MULT    EQUALS %x2 TAG c;           { T3 multiplier
    CONSTANT  BCT3MULT  EQUALS %x8 TAG c;           { Broadcast Circuit T3
                                                    {  multiplier
    CONSTANT  VER_LOWW  EQUALS %x0002 TAG c;        { Transport's version number
    CONSTANT  VER_HIB   EQUALS %x00 TAG c;          {       V2.0.0
    CONSTANT  HIORD     EQUALS %x000400AA TAG c;    { HIORD part of node address
    CONSTANT  RTR_LVL1  EQUALS %x2 TAG c;           { Level 1 router type code
    CONSTANT  RTR_LVL2  EQUALS %x1 TAG c;           { Level 2 router type code
    CONSTANT  END_NODE  EQUALS %x3 TAG c;           { Endnode type code
    CONSTANT  BCR_MID1  EQUALS %x030000AB TAG c;    { Broadcast circuit router's
    CONSTANT  BCR_MID2  EQUALS %X0 TAG c;           {  multicast ID
    CONSTANT  BCE_MID1  EQUALS %x040000AB TAG c;    { Broadcast circuit endnode's
    CONSTANT  BCE_MID2  EQUALS %X0 TAG c;           {  multicast ID
    CONSTANT  PRO_TYPE  EQUALS %x0360   TAG c;      { Transports protocol type

    {
    {
    {  Define message header sizes where applicable
    {
    {
    CONSTANT  hsz_data  EQUALS   21  TAG c;         { BC Endnode route-thru message

    END tr4msg;
AGGREGATE  tr4addr STRUCTURE  PREFIX tr4$;          { Node address qualifiers
    addr_dest BITFIELD  MASK LENGTH 10;             { Destination address field
    addr_area BITFIELD  MASK LENGTH 6;              { Area part of node address

    END tr4addr;
END_MODULE  $nspmsgdef ;
```

6 XWBDEF SDL FILES

```
                MODULE  $xwbdef;

{
{ Version:      'X-1'
{
{*
{*              XWBDEF.SDL      - Logical-link nonpaged control structures
{*
{*****************************************************************************
{*                                                                           *
{*  COPYRIGHT (c) 1978, 1980, 1982, 1984, 1986 BY                            *
{*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                   *
{*  ALL RIGHTS RESERVED.                                                     *
{*                                                                           *
{*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED    *
{*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE    *
{*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER    *
{*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY    *
{*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY    *
{*  TRANSFERRED.                                                             *
{*                                                                           *
{*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE    *
{*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT    *
{*  CORPORATION.                                                             *
{*                                                                           *
{*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS    *
{*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                  *
{*                                                                           *
{*                                                                           *
{*****************************************************************************
{*
{*
{*  AUTHOR:             Alan D. Eldridge        1-April-1982
{*
{*  MODIFIED BY:
{*
{*      X-1     PRB0377         Paul Beck       23-JUL-1985 17:02
{*              Add support for cluster node address, as follows:
{*                      Add XWB$V_STS_ALIAS to flag cluster node address outgoing.
{*
{*      V03-006 RNG0006         Rod Gamache     11-Jun-1984
{*              Add XWB$V_STS_NDC node counter access bit to XWB.
{*
{*      V03-005 ADE0001         A.Eldridge      21-Oct-1982
{*              Significant LSB redefinition.  Removed DLE stuff from XWB.
{*
{*
```

```
{
{  LOGICAL-LINK SUBCHANNEL BLOCK   (LSB)
{
{  The block is used to control the activity on a logical link
{  subchannel.  There are two subchannels: the DATA subchannel and the
{  INTERRUPT/LINK SERVICE subchannel.
{
{
{
AGGREGATE  lsb STRUCTURE PREFIX lsb$ ;  { Link Subchannel Block

{   KEEP LONGWORDS, WORDS, AND BYTES ALIGNED ON THEIR APPROPRIATE BOUNDARIES

    {
    {  Transmitter control variables
    {

    lux         WORD;           { Last segment # assigned to a segment
    lnx         WORD;           { Last segment # xmitted
    hxs         WORD;           { Highest segment # sendable
    har         WORD;           { Highest ACK # received
                                {
    haa         WORD;           { Highest ACK # acceptable
    x_req       BYTE;           { Flow control credits from remote receiver
    x_adj       BYTE;           { Packet window adjustment counter
                                {
    x_pktwnd    BYTE;           { Size of the transmit-packet-window
    x_cxbact    BYTE;           { Number of active transmit CXB's
    x_cxbquo    BYTE;           { Max total CXB's allowed
    x_cxbcnt    BYTE;           { Total CXB's both active on on the free queue
                                {
    x_pnd       ADDRESS TAG L;  { Listhead for xmt IRP's containing data
    x_irp       ADDRESS TAG L;  { Listhead for xmt IRP's with data moved to CXBs
    x_cxb       ADDRESS TAG L;  { Transmit CXB (message segments) listhead

    {
    {  Receiver control variables
    {

    r_irp       ADDRESS TAG L;  { Receive IRP listhead
    r_cxb       ADDRESS TAG L;  { Received CXB (message segments) listhead
                                {
    hnr         WORD;           { Highest numbered msg received and accepted
    hax         WORD;           { Highest ACK xmitted
                                {
    r_cxbcnt    BYTE;           { Number of CXB's in LSB list (unACK'ed)
    r_cxbquo    BYTE;           { Max rcv CXB's which can be buffered by NSP
                                { before some are passed to the Session Layer
    spare       BYTE;           { Spare, used for alignment

    {
```

```
{  Miscellaneous
{

sts             STRUCTURE TAG b;      { Status bits
    li          BITFIELD MASK;        { Set for LS/INT subchannel
    spare       BITFIELD LENGTH 4;    { skip over to coincide with NSP bits
    bom         BITFIELD MASK;        { Next seg to send has NSP$V_DATA_BOM set
    eom         BITFIELD MASK;        { Next seg to send has NSP$V_DATA_EOM set

    END sts;
cross           ADDRESS TAG L;    { Pointer to "cross-channel" LSB

#lsb_lng = .;                     { Length for use by XWB definition
END lsb;                          {
```

```
{
{  NETWORK WINDOW BLOCK  (XWB)  - Network version of a WCB
{
{  This control block serves as the Network Window Control Block, as such
{  its header section must look like a WCB.  The remainder of the
{  structure is Network specific.  There is one XWB per logical link.
{
{

AGGREGATE  xwb STRUCTURE PREFIX xwb$ ;
    {
    {
    {  The header portion of the block tracks the WCB format
    {
    {
    wlfl        ADDRESS TAG L;  { Window list forward link
    wlbl        ADDRESS TAG L;  { Window list backward link
    size        WORD;           { Bytes allocated for structure
    type        BYTE;           { Contains code identifying structure type
    access      BYTE;           { IO$_ACCESS control flags (see WCB definition)
    refcnt      WORD;           { Count of accessors of the window
    sts         STRUCTURE TAG w;{ Contains the miscellaneous status flags.
                                  { (DECnet specific)
        sts_tid     BITFIELD MASK;  { Set if XWB$W_TIM_ID is currently in use
        sts_tli     BITFIELD MASK;  { Set if XWB$W_TIM_ID used by LI subchannel
        sts_sol     BITFIELD MASK;  { Set if the XWB fork block is in use
        sts_dis     BITFIELD MASK;  { Set if synchronous disconnect is pending
                                    { awaiting RUN state run-down
        sts_con     BITFIELD MASK;  { Set until XWB enters the RUN state
        sts_tmo     BITFIELD MASK;  { Link timed out
        sts_rbp     BITFIELD MASK;  { Set if receiver is back-pressured off
        sts_ovf     BITFIELD MASK;  { Set if local rcvr is in "overflow" state
                                    {
        sts_dtnak   BITFIELD MASK;  { Set if next DATA ACK should be a NAK
        sts_linak   BITFIELD MASK;  { Set if next LS/INT ACK should be a NAK
        sts_astpnd  BITFIELD MASK;  { Special Kernel AST pending
        sts_astreq  BITFIELD MASK;  { Special Kernel AST requested
                                    {
        sts_ndc     BITFIELD MASK;  { Set if NODE COUNTER BLOCK is accessed
                                    {
        sts_alias   BITFIELD MASK;  { Set if connection uses alias address as
                                    { source.
        END sts;                    {
                                    {
    orgucb      ADDRESS TAG L;      { Original UCB address

    {
    {  The remainder of the block is DECnet specific.
    {

    fork        QUADWORD;           { Fork queue linkage
```

```
flg         STRUCTURE TAG w;     ! Flags to control message xmission.
    {
    !   Because an FFS instruction is used on the following to determine
    !   what to do next, the order of the bit definitions is critical.
    !
    flg_break   BITFIELD MASK;  ! Break the link
    flg_wbuf    BITFIELD MASK;  ! Wait for buffer availability
    flg_siack   BITFIELD MASK;  ! Send INT/LS ACK
    flg_sdack   BITFIELD MASK;  ! Send DATA ACK
    flg_sli     BITFIELD MASK;  ! Send INT/LS message
    !
    !   The next 2 bits are wait conditions on the DATA subchannel
    !
    flg_whgl    BITFIELD MASK;  ! Wait for HXS to become less than LUX
    flg_wbp     BITFIELD MASK;  ! Wait for backpressure to be relaxed
                                !
    flg_sdt     BITFIELD MASK;  ! Send a DATA message
    flg_scd     BITFIELD MASK;  ! Send Connect or Disconnect
    flg_clo     BITFIELD MASK;  ! Close the link and deallocate the XWB
    flg_wdat    BITFIELD MASK;  ! Waiting for transmit CXB useage to go
                                ! below quota
    !
    !  The next 4 flags are used for controlling when and how the next
    !  Link Service subchannel message is to be built.  They do not by
    !  themselves indicate that the message is to be transmitted.
    !
    !  Since the are scanned via a FFS instruction, they must be defined
    !  in order of their priority.
    !
    flg_tbpr    BITFIELD MASK;  ! Toggle back-pressure at remote end
    flg_iavl    BITFIELD MASK;  ! Build Interrupt message
    flg_sifl    BITFIELD MASK;  ! Build INT flow ctl msg
    flg_sdfl    BITFIELD MASK;  ! Build DATA flow ctl msg
                                !
    END flg;                    !
                                !
sta         BYTE;               ! Logical link states
    CONSTANT  (sta_clo,         ! Closed state
               sta_cis,         ! Connect Initiate Sending
               sta_car,         ! Connect Ack Received
               sta_cir,         ! Connect Initiate Received
               sta_ccs,         ! Connect Confirm Sending
               sta_run,         ! Run
               sta_dir,         ! Disconnect Initiate Received
               sta_dis,         ! Disconnect Initiate Sending
               numsta           ! Number of states (this works since
              )                 !  these constants start at "0").  This
          EQUALS 0 INCREMENT 1! value (8) allows a quadword per
          TAG c;                !  event in the state tables. Changing
                                !  it may be difficult.
fipl        BYTE;               ! Fork IPL level
fpc         ADDRESS TAG L;      ! Fork process PC value
```

```
fr3         LONGWORD;           { Fork process R3 value
fr4         LONGWORD;           { Fork process R4 value
link        ADDRESS TAG L;      { Link for XWB list
vcb         ADDRESS TAG L;      { Ptr to Volume Control Block (actually RCB)
pid         LONGWORD;           { I.D. of process given connect
path        WORD;               { Path (circuit number) over which to xmit.
                                { Zero implies choose the circuit number
                                { dynamically.  High byte is a sequence #
remnod      WORD;               { Network address of partner node
remlnk      WORD;               { Remote node's link address
loclnk      WORD;               { Local link address
locsiz      WORD;               { Maximum receive segment size
remsiz      WORD;               { Maximum transmit segment size
r_reason    WORD;               { Received disconnect reason
x_reason    WORD;               { Disconnect reason code to send to remote
tim_id      WORD;               { Identity of segment being timed
elapse      WORD;               { Seconds since the timer was last reset
tim_inact   WORD;               { Maximum inactivity interval, in seconds
delay       WORD;               { Estimated seconds between xmission and ACK
timer       WORD;               { Current timer value, in seconds
progress    WORD;               { Logical link confidence variable
retran      WORD;               { Maximum retransmissions before link is to
                                { be disconnected
dly_fact    WORD;               { Retransmission timer delay factor
dly_wght    WORD;               { Retransmission timer delay weight
pro         STRUCTURE TAG b;    { Partner's protocol capabilities
    pro_nfc     BITFIELD MASK;  { Partner's receiver uses "no-flow"
    pro_sfc     BITFIELD MASK;  { Partner's receiver uses "segment flow"
    pro_ph2     BITFIELD MASK;  { Parnter is Phase II (no timer support)
    pro_cca     BITFIELD MASK;  { Cross-channel ACKing allowed
    pro_nar     BITFIELD MASK;  { Can send "no ack request" flag to partner
                                {
    END pro;                    {
                                {
#x = 16;                        { Size DATA field
CONSTANT  data                  { Max size of DATA text field
            EQUALS #x TAG c;    {
data        BYTE;               { Count of bytes used in next field
data        CHARACTER           { Optional data to be sent in next connect
            LENGTH #x;          { or disconnect message
x_flw       BYTE;               { Transmit link service/flow control info
x_flwcnt    BYTE;               { Flow control count for next link service
                                { message to be transmitted
sp3         BYTE;               { Spare for alignment
#x = 16;                        { Size RID field
CONSTANT  rid                   { Max size of RID text field
            EQUALS #x TAG c;    {
rid         BYTE;               { Remote user (process, task, etc.) i.d.
rid         CHARACTER           {
            LENGTH #x;          {
irp_acc     LONGWORD;           { Ptr to IO$_ACCESS IRP (0 if none)
#ndc = 32;                      {
```

```
CONSTANT  ndc_lng                 { Define room for counter block
          EQUALS #ndc TAG c;      {
ndc       BYTE TAG  z             { Counter block area
          DIMENSION #ndc ;
{
{
{  The remainder of the structure is multiplexed depending upon the NSP
{  logical-link state, and depending upon whether the XWB is used for
{  logical-links or for Direct-line access.
{
{
comlng    UNION TAG c;            { XWB$C_COMLNG is the length of the common
                                  { XWB area above.
  run_blk STRUCTURE;              { While in the RUN state
    ""    CHARACTER LENGTH ((.+3)&(-4))-.;{ Force longword alignment
    dt    CHARACTER LENGTH #lsb_lng;      { DATA subchannel LSB
    ""    CHARACTER LENGTH ((.+3)&(-4))-.;{ Force longword alignment
    li    CHARACTER LENGTH #lsb_lng;      { LS/INT subchannel LSB
    dea_irp LONGWORD;                     { IO$_DEACCESS IRP
    END   run_blk;                        {
                                  {
  con_blk STRUCTURE;              { For all "connect" states the field usage
                                  { is as follows. Process names contain a
                                  { format field in the first byte, and have
                                  { the following formats:
                                  {
                                  { 0  followed by 1 byte non-zero object #
                                  { 1  followed by 1 byte of zero (object #)
                                  {    followed by a counted (max 16 bytes of
                                  {    test) object name
                                  { 2  followed by 1 byte of zero (object #)
                                  {    followed by 2 byte group code, 2 byte
                                  {    user code, and a counted (max 12 bytes
                                  {    of text object name
                                  {
    #y = 19;                      { Max size of process name field
    CONSTANT (lprnam,rprnam)      { Size of the following count and text
          EQUALS #y+1 TAG c;      { fields
    lprnam BYTE;                  { Count field for local process name
    lprnam CHARACTER LENGTH #y;{ Local process name
    rprnam BYTE;                  { Count field for remote process name
    rprnam CHARACTER LENGTH #y;{ Remote process name
                                  {
    #y = 63;                      { Max size of LOGIN compound string
    CONSTANT login EQUALS #y+1    { Size of LOGIN compound strings + count
                   TAG c;         { field
    login  BYTE;                  { Count of the following composite field
                                  {  (mininum value is 3)
    login  CHARACTER LENGTH #y;{ Three concatenated counted account strings
                                  { (min value 3 consecutive 0's)
    icb    ADDRESS TAG l;         { Pointer to Internal Connect Block
    ci_path WORD;                 { Path i.d. over which Connect Initiate
```

```
                                        ( was received.
    #y = .;                             ( Length to this point
    CONSTANT conlng EQUALS #y           (
                    TAG c;              (
    END con_blk;                        (
  END conlng;                           (

"" CHARACTER LENGTH ((.+7)&(-8))-.; ( Force quadword alignment
free_cxb    QUADWORD;                   ( Listhead of free CXB's
                                        (
END xwb;                                (


END_MODULE  $xwbdef;
```

7 PSIUSR SDL FILES

```
{
{ Version:      'V04-000'
{
{*****************************************************************************
{*                                                                           *
{*  COPYRIGHT (c) 1978, 1980, 1982, 1984, 1986  BY                           *
{*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                   *
{*  ALL RIGHTS RESERVED.                                                     *
{*                                                                           *
{*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED    *
{*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE    *
{*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER    *
{*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY    *
{*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY    *
{*  TRANSFERRED.                                                             *
{*                                                                           *
{*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE    *
{*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT    *
{*  CORPORATION.                                                             *
{*                                                                           *
{*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS    *
{*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                  *
{*                                                                           *
{*                                                                           *
{*****************************************************************************

{++
{
{ Facility:
{
{       VAX-11 PSI -- VMS
{
{ Abstract:
{
{       This file contains all definitions which must eventually be
{       inserted in the VMS system libraries - SYS$LIBRARY:LIB and
{       SYS$LIBRARY:STARLET. They should be collected in this file
{       for the time being.
{
{ Environment:
{
{       SDL processing - BLISS/MACRO Libraries.
{
{ Author:
{
{       Richard J. Merewood,    Date: 09-Sep-1980
{
{
{ Edit  Author          Date            Reason
{ ----  ------          ----            ------
{
```

```
{ 01     AM Palka          09-Sep-80       Add symbols for QIO and Level 3.
{ 02     AM Palka          26-Sep-80       Correct definition of PSI$C_NCB...
{ 03     AM Palka          29-Sep-80       Add more NCB definitions.
{ 04     Barry Scott        7-Nov-80       Add DYN$C_xxx symbols.
{ 05     AM Palka          24-Nov-80       Add PSI status codes.
{ 06     Barry A. Scott    25-Nov-80       Add DT$_X25.
{ 07     Andrew Palka      25-Nov-80       Change spelling of some NCB items.
{ 08     R.J. Merewood     03-Dec-80       Clean up and add NFB defintions.
{ 09     Fiona Nicholson   09-Dec-80       Add X29 device type
{ 10     AM Palka          10-Dec-80       change PSI$C_NCB values
{ 11     AM Palka          14-Jan-81       change SS$_CLEARED/RESET
{ 12     Andrew Palka      16-Jan-81       add more NCB definitions (for internal
{                                          L3CS use), Remove PSI$C_NCB_BUG
{ 13     R.J. Merewood     22-Jan-81       Add PSI$C_NTD_... codes.
{ 14     Andrew Palka      22-Jan-81       Add PSI$C_NCB_LOCFACR,
{                                          Remove PSI$C_NCB_ACCRC
{ 15     Andrew Palka      26-Jan-81       Add PSI$K_SETREADPAD
{ 16     Andrew Palka       4-Feb-81       Remove PSI$C_NCB_FRNFAC and
{                                          PSI$C_NCB_REMNET.
{                                          Add PSI$C_NCB_LOCNET
{ 17     Barry A. Scott    9-Feb-81        Add NFB$C_READRSP for XUT interface
{ 18     R.J. Merewood     20-Feb-81       Change values as required by VMS group
{ 19     Barry A. Scott     2-Mar-1981     Add NFB$C_?ETTRASTA for trace.
{ 20     Andrew Palka       5-Mar-1981     Remove SS$_UNSOLICIT
{ 21     Andrew Palka       5-Mar-1981     ADD PSI$C_ERR_BAD_PVCNAME
{ 22     Andrew Palka       9-Mar-1981     Replace SS$_UNSOLICIT ( Not used, but
{                                          space for it has been reserved)
{ 23     Andrew Palka      12-Mar-1981     Add PSI$C_RESTART
{ 24     Barry A. Scott     6-Apr-1981     Add NFB$C_TESTLINE for the LOOP LINE
{                                          function
{ 25     R.J. Merewood     13-Apr-81       Add new NFB codes.
{ 26     Fiona Nicholson   22-Feb-81       Remove temporary definitions, as they
{                                          are in the system library now.
{--
```

```
{+
{ DT:
{       These are new device types for PSI.
{-
module $DTTMPDEF;

                                                        /* Start at 99 BS006
constant(
      X25                                               /* X.25 Network Device
    , X29                                               /* X.29 Device FN009
    , DUP_BOP                                           /* DUP-11 Bit-orientated protocol RM008
    ) equals 99 increment 1  prefix DT tag $;           /*


end_module $DTTMPDEF;
```

```
module $NFBTMPDEF;
/*+
/* NFB:
/*        These are new Network Function Block codes used by PSI.              RM008
/*-

                                                        /* Start at unused value after NFB$C_FC_MAX
constant(
      SETCCTSTA                                         /* Set circuit state
    , GETCCTSTA                                         /* Get circuit state
    , GETCCTCTR                                         /* Get circuit counters
    , CLRCCTCTR                                         /* Clear circuit counters
    , SETMODSTA                                         /* Set module state
    , GETMODSTA                                         /* Get module state
    , GETMODCTR                                         /* Get module counters
    , CLRMODCTR                                         /* Clear module counters
    , READRSP                                           /* Read response from ACPCONTROL BS017
    , SETTRASTA                                         /* set trace state BS019
    , GETTRASTA                                         /* Get trace state BS019
    , TESTLINE                                          /* Test line function BS024
    , SETOBJECT                                         /* Update object database RM025
    , SHOOBJECT                                         /* Display object database RM025
    ) equals (37+1) increment 1 prefix NFB tag $C;

end_module $NFBTMPDEF;
```

```
module $PSIDEF;
/*+
/* PSI:
/*        These are Network Connect Block codes and various status codes.
/*-


constant(
      NCB_NULL                                  /* Network connect null data type
    , NCB_REMDTE                                /* Remote DTE address
    , NCB_REMSUBADR                             /* Remote DTE sub-address
    , NCB_LOCDTE                                /* Local DTE address
    , NCB_LOCSUBADR                             /* Local DTE subaddress
    , NCB_USERDATA                              /* Outgoing call data
    , NCB_RESPDATA                              /* Response data field
    , NCB_OBJNAME                               /* Process to handle call
    , NCB_ENDTOEND                              /* Outgoing end to end acknowledgment
    , NCB_ENDTOENDR                             /* Call requires end to end acknowledgment
    , NCB_GATEWAY                               /* Gateway name
    , NCB_ICI                                   /* Incoming call identifier
    , NCB_REVCHG                                /* Reverse charge for this call
    , NCB_LOCFACR                               /* Facilities for call accept
    , NCB_FSEL                                  /* Fast select for outgoing call
    , NCB_FSEL_RES                              /* Restricted response fast select
    , NCB_THRUCLS                               /* Throughput class
    , NCB_CUG                                   /* Closed user group name
    , NCB_DIAGCODE                              /* Clear 8 bit diagnostics code
    , NCB_CAUSE                                 /* Clear 8 bit cause code
    , NCB_REASON                                /* Clear 8 bit reason code
    , NCB_PKTSIZE                               /* Packet size-must be power of 2
    , NCB_WINSIZE                               /* Window size
    , NCB_LOCFAC                                /* Local facilities
    , NCB_PVCNAM                                /* PVC name

/*+
/* The following codes are mainly for internal use.
/* They may not normally be used by a user program
/* (require PHYSIO and SYSPRV privileges)
/*-

    , NCB_LINE                                  /* Line name
    , NCB_PVCSET                                /* Set up PVC
    , NCB_LCN                                   /* Select LCN
    , NCB_CUGN                                  /* Closed user group number
    , NCB_CUGNB                                 /* Bilateral closed user group number
    , NCB_LOCNET                                /* Local network name
    ) equals 0 increment 1 prefix PSI tag $C;


/*+
/* Item List Codes for Network Process Declaration                RM013
/*-
```

```
                                                        /* Start with 1 RM013
constant(
      NTD_ACCLVL                                        /* Access level (X.25 or X.29) RM013
    , NTD_SALO                                          /* DTE subaddress range low end RM013
    , NTD_SAHI                                          /* DTE subaddress range high end RM013
    , NTD_REMDTE                                        /* Remote DTE address RM013
    , NTD_USRGRP                                        /* CUG/BCUG Name RM013
    , NTD_USRDATA                                       /* User data field RM013
    , NTD_DATMSK                                        /* User data mask RM013
    ) equals 1 increment 1 prefix PSI tag $C;


/*+
/* Special function codes for IO$_NETCONTROL
/*-

constant INTERRUPT      equals 1  prefix PSI tag $C;    /* Interrupt NETCONTROL functn code
constant INTERRUPT      equals 1  prefix PSI tag $K;    /* Interrupt NETCONTROL funct code
constant INTACK equals 2  prefix PSI tag $C;            /* Interrupt acknowledge NETCONTROL
constant INTACK equals 2  prefix PSI tag $K;            /* Interrupt acknowledge NETCONTROL
constant RESET  equals 3  prefix PSI tag $C;            /* Reset NETCONTROL function Code
constant RESET  equals 3  prefix PSI tag $K;            /* Reset NETCONTROL function Code
constant SETPAD equals 4  prefix PSI tag $C;            /* Set X.29 PAD parameters FN009
constant SETPAD equals 4  prefix PSI tag $K;            /* Set X.29 PAD parameters FN009
constant READPAD        equals 5  prefix PSI tag $C;    /* Read X.29 PAD parameters FN009
constant READPAD        equals 5  prefix PSI tag $K;    /* Read X.29 PAD parameters FN009
constant SETREADPAD     equals 6  prefix PSI tag $C;    /* Set/Read X.29 PAD parameters AP015
constant SETREADPAD     equals 6  prefix PSI tag $K;    /* Set/Read X.29 PAD parameters AP015
constant RESTART        equals 7  prefix PSI tag $C;    /* Acknowlege Restart on PVC AP023
constant RESTART        equals 7  prefix PSI tag $K;    /* Acknowlege Restart on PVC AP023

/*+
/* Clear/Reset/Restart Cause Codes from Level 3
/*-


                                                        /* Start at 1 RM008
constant(
      L3_LOCAL                                          /* Host requested
    , L3_NETWRK                                         /* Network initiated
    , L3_NETERR                                         /* Network protocol error
    , L3_LOCERR                                         /* Local protocol error
    , L3_LNKRST                                         /* Link reset
    , L3_LNKDWN                                         /* Link down
    , L3_LNKUP                                          /* Link up
    , L3_LNKRRT                                         /* Link restarted
    , L3_LOCMGT                                         /* Network management function
    , L3_CALCOL                                         /* Call collision
    , L3_NETTIM                                         /* Timeout on network
    ) equals 1 increment 1 prefix PSI tag $C;


/*+
```

```
/* I/O Status Returns
/*-

aggregate PSIDEF  union fill prefix PSI$;
    PSIDEF_BITS0 structure fill;
        MOREDATA bitfield mask;                 /* More data follows (M-bit)
        QUALIFIED bitfield mask;                /* Qualified sub-channel (Q-bit)
    end PSIDEF_BITS0;

    PSIDEF_BITS1 structure fill;
        STS_PVC bitfield mask;                  /* PVC setup MUST BE LOW ORDER BIT
        STS_LOCDTELNG bitfield mask;            /* Local dte adress too long - truncated
        STS_REMDTELNG bitfield mask;            /* Remote dte adress too long - truncated
        STS_USERLNG bitfield mask;              /* Too much user data supplied - truncated
        STS_WINBAD bitfield mask;               /* Invalid window size - nearest valid chosen
        STS_PKTBAD bitfield mask;               /* Invalid packet size - nearest valid chosen
        STS_THRBAD bitfield mask;               /* Invalid throughput class - nearest valid chosen
    end PSIDEF_BITS1;

                                                /* Error codes start at 0 AP005
    constant(
          ERR_UNKNOWN                           /* Unspecified internal error
        , ERR_FACLNG                            /* Facilities too long
        , ERR_INVITEM                           /* Invalid item code
        , ERR_CONFLICT                          /* Conflicting items specified
        , ERR_BADPARM                           /* Bad parameter specified
        , ERR_NOTRANS                           /* No translation for this name (e.g. unknown user group)
        , ERR_RECURLMT                          /* Recursion limit reached
        , ERR_INVNUM                            /* Invalid ASCII number
        , ERR_NOICI                             /* No internal call identifier specified
        , ERR_MANYICI                           /* More than one internal call identifier given
        , ERR_NOTIMP                            /* A feature that is not yet implemented was requested
        , ERR_NOLINES                           /* No line is available on which to make the call
        , ERR_NOSUCHLINE                        /* The specified line is not known
        , ERR_NOSUCHPVC                         /* The specified PVC is not known
        , ERR_NOSUCHNET                         /* The specified network is not known
        , ERR_NOLOCAL                           /* The ACP has run out of local workspace memory
        , ERR_NONONPAG                          /* There is insufficient free non-paged pool
        , ERR_NOL3                              /* Internal error
        , ERR_BADNAME                           /* Bad counted string parameter
        , ERR_L3ERR                             /* Error returned from level 3
        , ERR_PVCALRACC                         /* PVC already accessed
        , ERR_BAD_PVCNAME                       /* Accessing PVC on wrong channel
        ) equals 0 increment 1 prefix PSI tag $C;

end PSIDEF;

end_module $PSIDEF;

SS
```