

Ps
--
NE

NE

NE

NE

SR

NNN		NNN	EEEEEEEEEEEEEEEE	TTTTTTTTTTTTTTTT	AAAAAAAAAA		CCCCCCCCCCCC	PPPPPPPPPP	
NNN		NNN	EEEEEEEEEEEEEEEE	TTTTTTTTTTTTTTTT	AAAAAAAAAA		CCCCCCCCCCCC	PPPPPPPPPP	
NNN		NNN	EEEEEEEEEEEEEEEE	TTTTTTTTTTTTTTTT	AAAAAAAAAA		CCCCCCCCCCCC	PPPPPPPPPP	
NNN		NNN	EEE	TTT	AAA	AAA	CCC	PPP	PPP
NNN		NNN	EEE	TTT	AAA	AAA	CCC	FPP	PPP
NNN		NNN	EEE	TTT	AAA	AAA	CCC	PPP	PPP
NNNNNN		NNN	EEE	TTT	AAA	AAA	CCC	PPP	PPP
NNNNNN		NNN	EEE	TTT	AAA	AAA	CCC	PPP	PPP
NNNNNN		NNN	EEE	TTT	AAA	AAA	CCC	PPP	PPP
NNN	NNN	NNN	EEEEEEEEEEEE	TTT	AAA	AAA	CCC	PPPPPPPPPP	PPP
NNN	NNN	NNN	EEEEEEEEEEEE	TTT	AAA	AAA	CCC	PPPPPPPPPP	PPP
NNN	NNN	NNN	EEEEEEEEEEEE	TTT	AAA	AAA	CCC	PPPPPPPPPP	PPP
NNN		NNNNNN	EEE	TTT	AAAAA	AAA	CCC	PPP	PPP
NNN		NNNNNN	EEE	TTT	AAAAA	AAA	CCC	PPP	PPP
NNN		NNNNNN	EEE	TTT	AAAAA	AAA	CCC	PPP	PPP
NNN		NNN	EEE	TTT	AAA	AAA	CCC	PPP	PPP
NNN		NNN	EEE	TTT	AAA	AAA	CCC	PPP	PPP
NNN		NNN	EEE	TTT	AAA	AAA	CCC	PPP	PPP
NNN		NNN	EEEEEEEEEEEE	TTT	AAA	AAA	CCC	PPP	PPP
NNN		NNN	EEEEEEEEEEEE	TTT	AAA	AAA	CCCCCCCCCCCC	PPP	PPP
NNN		NNN	EEEEEEEEEEEE	TTT	AAA	AAA	CCCCCCCCCCCC	PPP	PPP
NNN		NNN	EEEEEEEEEEEE	TTT	AAA	AAA	CCCCCCCCCCCC	PPP	PPP

```

NN      NN      SSSSSSSS  PPPPPPPP  MM      MM      SSSSSSSS  GGGGGGGG  DDDDDDDD  EEEEEEEEEE  FFFFFFFFFF
NN      NN      SSSSSSSS  PPPPPPPP  MM      MM      SSSSSSSS  GGGGGGGG  DDDDDDDD  EEEEEEEEEE  FFFFFFFFFF
NN      NN      SS        PP        PP  MMMM  MMMM  SS        GG        DD        DD  EE        FF
NN      NN      SS        PP        PP  MMMM  MMMM  SS        GG        DD        DD  EE        FF
NNNN    NN      SS        PP        PP  MM  MM  MM  SS        GG        DD        DD  EE        FF
NNNN    NN      SS        PP        PP  MM  MM  MM  SS        GG        DD        DD  EE        FF
NN  NN  NN      SSSSSS  PPPPPPPP  MM      MM      SSSSSS  GG        GG        DD        DD  EEEEEEEE  FFFFFFFF
NN  NN  NN      SSSSSS  PPPPPPPP  MM      MM      SSSSSS  GG        GG        DD        DD  EEEEEEEE  FFFFFFFF
NN      NNNN      SS        PP        MM      MM      SS        GG  GGGGGG  DD        DD  EE        FF
NN      NNNN      SS        PP        MM      MM      SS        GG  GGGGGG  DD        DD  EE        FF
NN      NN      SS        PP        MM      MM      SS        GG        GG        DD        DD  EE        FF
NN      NN      SS        PP        MM      MM      SS        GG        GG        DD        DD  EE        FF
NN      NN      SSSSSSSS  PP        MM      MM      SSSSSSSS  GGGGGG  DDDDDDDD  EEEEEEEEEE  FF
NN      NN      SSSSSSSS  PP        MM      MM      SSSSSSSS  GGGGGG  DDDDDDDD  EEEEEEEEEE  FF

```

```

SSSSSSSS  DDDDDDDD  LL
SSSSSSSS  DDDDDDDD  LL
SS        DD        DD  LL
SS        DD        DD  LL
SS        DD        DD  LL
SS        DD        DD  LL
SSSSSS    DD        DD  LL
SSSSSS    DD        DD  LL
SS        DD        DD  LL
SS        DD        DD  LL
SS        DD        DD  LL
SS        DD        DD  LL
SSSSSSSS  DDDDDDDD  LLLLLLLLLL
SSSSSSSS  DDDDDDDD  LLLLLLLLLL

```

P
C
E
M

MODULE \$nspmsgdef

NSPMSGDEF.SDL - NSP and Transport Message Definitions
Version 'V04-000'

```

*****
(*  COPYRIGHT (c) 1978, 1980, 1982, 1984 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:

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-C03 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><2b_SRV><2b_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
{ <2b_ACK    ::= <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
{ <1b_SRV>   ::= <00000101>                            if segment flow control
{ <1b_SRV>   ::= <00001001>                            if message flow control
{ <1b_INFO>  ::= <00000001>                            if NSP V3.1
{ <1b_INFO>  ::= <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
{ <NAME>    ::= <1k_1><1k_0><i16_desc>
{ <NAME>    ::= <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_d1    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;
```

```
qual_msg STRUCTURE TAG $$;
msg_sp1  BITFIELD LENGTH 4;
msg_li   BITFIELD MASK;
msg_int  BITFIELD MASK;
dstlnk   WORD;
srclnk   WORD;
END qual_msg;
{ Miscellaneous message fields
{ Misc. MSG field qualifiers
{ Skip constant part of field
{ Set if not DATA subchannel msg
{ Set if INT rather than LS msg
{ Destination link address
{ Source link address
{
```

```
qual_data STRUCTURE TAG $$;
data_sp   BITFIELD LENGTH 5;
data_bom  BITFIELD MASK;
data_eom  BITFIELD MASK;
data_ovfw BITFIELD MASK;
END qual_data;
{ Qualifiers to NSP$C_MSG_DATA
{ Skip constant part of field
{ Set if first seg in message
{ Set if last seg in message
{ Set if no further segs could
{ be buffered without overflow
{ of the session ctl buffer -
{ INTERNAL ONLY, not in NSP
{
```

```
qual_ack STRUCTURE TAG $$;
ack_num  BITFIELD MASK LENGTH 12;
ack_nak  BITFIELD MASK;
ack_sp2  BITFIELD LENGTH 2;
ack_valid BITFIELD MASK;
{ Qualifiers to all ACK fields
{ Seg number being ACK'd
{ Set if this is a NAK
{ Skip reserved field
{ Set to identify field as an
{
```

```

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

```

```
flw_lisub BITFIELD MASK LENGTH 1; { Flow control count is for the
flw_sp1 BITFIELD MASK LENGTH 1; { Link Service/Interrupt, and
flw_inuse BITFIELD MASK LENGTH 1; { not the DATA, subchannel
flw_int BITFIELD MASK LENGTH 1; { Spare
                                { XWBSB_X_FLW is in use
                                { XWBSB_X_FLW describes an
                                { "Interrupt", not a "Link-
                                { Service", message
                                { Spare
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
  msg_rth   BITFIELD  MASK LENGTH 1;            { - clear on all other messages
                                                    { Set if a Phase II or III
                                                    { route-header, clear if a
                                                    { Phase II control message
                                                    {

  end qual_msg;

  qual_rtrlg 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-serder"
  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_rtrlg;

  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 i001>   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 $$;
```

```
{
{ 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
{
```

```
{ 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
```

```
{
{ Define qualifiers to message codes
```

```
{
qual UNION; { Miscellaneous message fields
```

```
qual_rtflg STRUCTURE TAG $$; { Long Route-header
{ qualifiers
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 qual_rtflg;
```

```
qual_sclass STRUCTURE TAG $$; { Long format SERVICE CLASS
{ qualifiers
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 qual_sclass;
```

```
qual_addr STRUCTURE TAG $$; { Node address qualifiers
addr_dest BITFIELD MASK LENGTH 10; { Destination address field
addr_area BITFIELD MASK LENGTH 6; { Area part of node address
END qual_addr;
```

```
END qual;
```

```
END tr4msg;
```

```
END_MODULE $nspmsgdef ;
```


