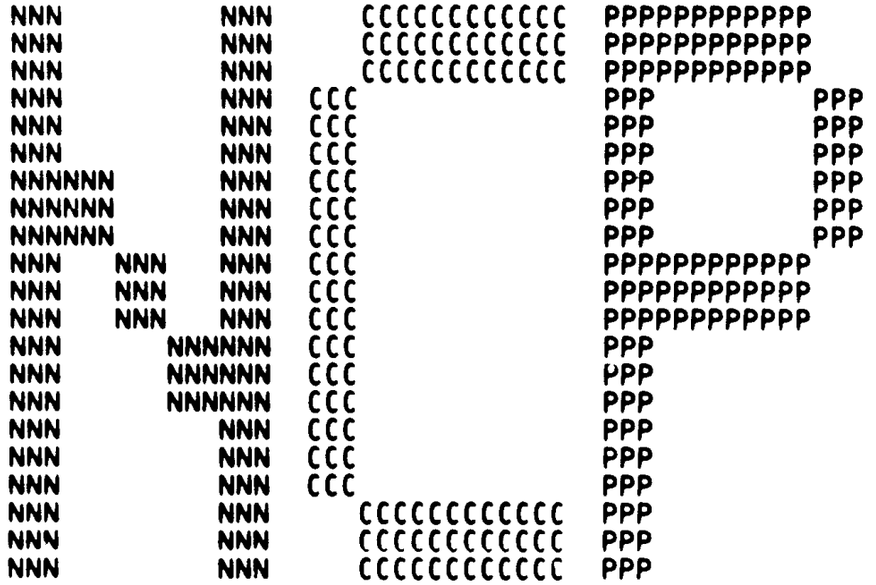
NNN NNN CCCCCCCCCCC PPPPPPPPPPPP
NNN NNN CCCCCCCCCCC PPPPPPPPPPPP
NNN NNN CCCCCCCCCCC PPPPPPPPPPPP
NNN NNN CCC PPP PPP
NNN NNN CCC PPP PPP
NNN NNN CCC PPP PPP
NNNNNN NNN CCC PPP PPP
NNNNNN NNN CCC PPP PPP
NNNNNN NNN CCC PPP PPP
NNN NNN NNN CCC PPPPPPPPPPPP
NNN NNN NNN CCC PPPPPPPPPPPP
NNN NNN NNN CCC PPPPPPPPPPPP
NNN NNNNNN CCC PPP
NNN NNNNNN CCC PPP
NNN NNNNNN CCC PPP
NNN NNN CCC PPP
NNN NNN CCC PPP
NNN NNN CCC PPP
NNN NNN CCCCCCCCCCC PPP
NNN NNN CCCCCCCCCCC PPP
NNN NNN CCCCCCCCCCC PPP

```
NN       NN    CCCCCCCC  PPPPPPPP    LL            IIIIII    BBBBBBBB    RRRRRRRR    YY       YY
NN       NN    CCCCCCCC  PPPPPPPP    LL            IIIIII    BBBBBBBB    RRRRRRRR    YY       YY
NN       NN  CC          PP      PP  LL              II      BB      BB  RR      RR  YY       YY
NN       NN  CC          PP      PP  LL              II      BB      BB  RR      RR  YY       YY
NNNN     NN  CC          PP      PP  LL              II      BB      BB  RR      RR    YY   YY
NNNN     NN  CC          PP      PP  LL              II      BB      BB  RR      RR    YY   YY
NN  NN   NN  CC          PPPPPPPP    LL              II      BBBBBBBB    RRRRRRRR        YY
NN  NN   NN  CC          PPPPPPPP    LL              II      BBBBBBBB    RRRRRRRR        YY
NN    NNNN  CC          PP          LL              II      BB      BB  RR  RR          YY
NN    NNNN  CC          PP          LL              II      BB      BB  RR  RR          YY
NN      NN  CC          PP          LL              II      BB      BB  RR    RR        YY          ....
NN      NN  CC          PP          LL              II      BB      BB  RR    RR        YY          ....
NN      NN    CCCCCCCC  PP          LLLLLLLLLL    IIIIII    BBBBBBBB    RR      RR      YY          ....
NN      NN    CCCCCCCC  PP          LLLLLLLLLL    IIIIII    BBBBBBBB    RR      RR      YY          ....

BBBBBBBB      333333      222222
BBBBBBBB      333333      222222
BB      BB  33      33  22      22
BB      BB  33      33  22      22
BB      BB          33          22
BB      BB          33          22
BBBBBBBB        33          22
BBBBBBBB        33          22
BB      BB          33      22
BB      BB          33      22
BB      BB  33      33    22
BB      BB  33      33    22
BBBBBBBB      333333    2222222222
BBBBBBBB      333333    2222222222
```

```
%TITLE 'NCPLIBRY Symbol Definition Library'
!MODULE NCPLIBRY (IDENT = 'V04-000') =
!BEGIN
```



```
!
!++
! FACILITY:     NCP Network Control Program (NCP)
!
! ABSTRACT:
!
!       NCP Library of common definitions
!
! ENVIRONMENT:  VAX/VMS Operating System
!
! AUTHOR:       Darrell Duffy   , CREATION DATE: 28-August-1979
!
! MODIFIED BY:
!
!       V03-031 PRD0112         Paul R. DeStefano       31-Jul-1984
!               Allow node address and executive node address of 0.
!
!       V03-030 PRD0104         Paul R. DeStefano       18-Jul-1984
!               Allow underscores ("_") to be included in group,
!               network, and destination names.
!
!       V03-029 PRD0050         Paul R. DeStefano       05-Feb-1984
!               Added state expression to parse OBJECT parameter as
!               a number.
!               Changed ACT$GL_NODADR_Q to more general name ACT$GL_ADR_Q.
!
!       V03-028 RPG0028         Bob Grosso              10-Jun-1983
!               Add service device UNA.
```

```
V03-027 RPG0027         Bob Grosso              22-Mar-1983
        Turn off BLANKS after termination of state expression
        macro to parse NI addresses.

V03-026 RPG0026         Bob Grosso              16-Mar-1983
        Update NCP version number to IV.
        Complete state expression macro to parse NI addresses.

V03-025 RPG0025         Bob Grosso              10-Mar-1983
        Add state expression macro to parse NI addresses.

V03-024 RPG0024         Bob Grosso              25-Feb-1983
        Remove syntax checking for NODE id and correct
        parsing of circuit names.
        Note, this packet cannot be backed off without taking
        RPG0023 with it.

V03-023 RPG0023         Bob Grosso              18-Feb-1983
        Remove syntax checking for line-id and circuit-id.
        Change High range for node adr from 255 to 1023.
        Add high and low for LINE BFS.
        Add high and low for NODE FBS and SBS.

V03-022 RPG0022         Bob Grosso              20-Oct-1982
        Allow '$' and '_' in object names.  Allow 12 character
        object names.
        Have SE_NODE_ADR flag Area present in node address.

V03-021 RPG0021         Bob Grosso              23-Sep-1982
        Parse for node area.
        Range for Module Console RTR

V03-020 RPG0020         Bob Grosso              15-Sep-1982
        Increase tracepoint name length to 30 from 16.

V03-019 RPG0019         Bob Grosso              03-Sep-1982
        Add range for LIN RTT.
        Change range for MTR BSZ.
        Add SEM_HEX_NUM and LEN_HEX_NUM to parse hex numbers.

V03-018 TMH0018         Tim Halvorsen           16-Aug-1982
        Change tracepoint name parsing to accept a string of
        any size, including periods as legal characters.

V03-017 RPG0017         Bob Grosso              03-Aug-82
        Add range for Module X25-Protocol MCI.
        Change QUERY_STATES_S to allow different ALL prompt
        strings to support sub-databases.

V03-016 RPG0016         Bob Grosso              23-Jul-82
        Support X25-Trace with subexpression for tracepoint names,
        SEM_TRCPNT_NAME.

V015    RPG0015         Bob Grosso              14-Jul-82
        Add NI support in Set Node by adding range values for
```

AMC, AMH, BRT, MAR, MBE, MBR.

V014 RPG0014 Bob Grosso 15-Jun-82
Add MODULE parameter table.
Add macro QUERY_STATES_S patterned after QUERY_STATES
to permit alternate prompting within entities without
having multiply defined states.
Add Subexpression and constant for channels lists.

V013 TMH0013 Tim Halvorsen 05-Apr-1982
Add ACT$TESTLONG action routine to ACT_DFN macro.
Allow numeric characters in line/circuit mnemonic.
Add circuit MRT and RPR ranges.
Allow any characters following initial dash after
line/circuit mnemonic (such as X25-CHICAGO).

V012 TMH0012 Tim Halvorsen 08-Jan-1982
Remove TMH0005, thus restoring RETRANSMIT TIMER
to a line parameter, which is what NM V3.0 finally
came up with.

V011 TMH0011 Tim Halvorsen 31-Dec-1981
Add DMF as a MOP service device.

V010 TMH0010 Tim Halvorsen 25-Nov-1981
Allow embedded spaces in filespecs as long as they
appear in double quotas (access control string).
This allows access control strings to be specified
in the filespec after the TO clause in the SHOW command.

V009 TMH0009 Tim Halvorsen 22-Oct-1981
Fix HEX_PSW sub-expression so that blank which terminates
hex password string does not get included in string.

V008 LMK0001 Len Kawell 19-Sep-1981
Change NICE version to 3.0.

V007 TMH0007 Tim Halvorsen 28-Aug-1981
Add macro to parse link ID

V006 TMH0006 Tim Halvorsen 15-Aug-1981
Add DMP, DMV and DPV service devices.
Add EXECUTOR PIPELINE QUOTA range.

V005 TMH0005 Tim Halvorsen 05-Aug-1981
Change RETRANSMIT TIMER to a circuit parameter
from a line parameter.

V004 TMH0004 Tim Halvorsen 07-Jul-1981
Rename maximum blocks to maximum transmits
Allow dashs in circuit names.

V003 TMH0003 Tim Halvorsen 11-Jun-1981
Add ranges for new V2.2 circuit parameters.
Remove obsolete line polling parameters.
Change NCP version number to 2.2.0

```
!
!       V02-002 LMK0001 Len Kawell      18-Dec-1980
!               Fix file-id parsing.
!--
```

```
%SBTTL  'Definitions'

!
! TABLE OF CONTENTS:
!

!
! MACROS:
!

!       Program Identification String

MACRO
    PRG_ID_STR =
        %STRING ('V3.00 ')
        %
        ,

!       Build a cr lf pair in a string

    CRLF =
        %CHAR (13, 10)
        %
        ;

!
! $FAB_DEV - a macro which defines a single FAB$L_DEV bit.
!       $FAB_DEV( bit_name )
!       where:
!           "bit_name" is a 3-character device bit name
!

MACRO
    $FAB_DEV( BIT_NAME ) =
        FAB$DEV( FAB$L_DEV, %NAME('DEV$V_',BIT_NAME) ) %,

    FAB$DEV( FAB_BYTE, FAB_BIT, FAB_SIZE, FAB_SIGN, DEV_DISP,
                DEV_BIT, DEV_SIZE, DEV_SIGN ) =
        FAB_BYTE, DEV_BIT, DEV_SIZE, DEV_SIGN %
    ;
```

```
!
!       Create a descriptor for a constant string
!

MACRO
    ASCID [] =
        (UPLIT BYTE                     ! Use byte alignment to save space
            (
            LONG (                      ! Parts must be longwords
                %CHARCOUNT( %STRING( %REMAINING)),
                UPLIT( %STRING( %REMAINING))
                )
            )
        )%
    ;

!
!       Create pointer to counted string
!

MACRO
    ASCIC [] =
        ( UPLIT BYTE (%ASCIC %STRING (%REMAINING) ) )
        %;

!
! Structure declarations used for system defined structures to
! save typing.  These structures are byte sized.
!       (Thanks to A. Goldstein)
!

STRUCTURE
        BBLOCK [O, P, S, E; N] =
            [N]
            (BBLOCK+O)<P,S,E>,

        BBLOCKVECTOR [I, O, P, S, E; N, BS] =
            [N*BS]
            ((BBLOCKVECTOR+I*BS)+O)<P,S,E>
        ;


!
!       Concatenate text to the control string
!

MACRO
        ADDSTR (TXT) =
                NCP$ADDSTR (ASCIC (TXT), NCP$GQ_CTRDSC)
        %;

!
!       Add an entry to the fao list
!
```

```
MACRO
        ADDFAO (ITEM) =
            NCP$ADDFAO (ITEM)
        %;
```

```
%SBTTL  'Macros to Build State Tables'

!
!       Macros to help build state tables
!
!
!   For the following macros:
!
!   CLS         Code for the sub-command
!   NAM         Parameter name
!
!
!   All state names have the form ST_CLS_...
!   There are two types of states, prompt and process. Prompt states
!   sequence the prompts for parameters.  Process states allow any
!   parameter in any order.
!
!
!       Build a sequence of prompt states
!       A prompt is printed and then it is parsed.  No answer is required
!       and if none is given the next prompt is issued.  If the response is
!       "_DONE" then the remainder of the prompts are skipped and the
!       function is performed.
!

MACRO
    PROMPT_STATES (CLS) [NAM] =

$STATE  (%NAME ('ST_', CLS, '_PMT_', NAM),
        (TPA$_LAMBDA, ,
            ACT$PRMPT, , , %NAME ('PMT$G_', CLS, '_', NAM) )
        );

$STATE  (,
        (TPA$_SYMBOL, %NAME ('ST_', CLS, '_DOIT'), ACT$PMTDONEQ ),
        ( (%NAME ('ST_', CLS, '_', NAM) ) ),
        (TPA$_EOS),
        (TPA$_LAMBDA, %NAME ('ST_', CLS, '_PMT_', NAM),
                ACT$SIGNAL, , , NCP$_INVVAL)
        );
    %;
```

```
!
!       Build a pair of states to accomplish command prompting
!
!       The idea is to cause prompting only if the state is entered
!       with TPA$_EOS true.  If prompting is true, then the state should
!       loop until either a transition is satisfied or the command is
!       canceled.  This is done by using ACT$PMT_ON and _OFF to remember
!       the state of prompting and ACT$PMT_Q to act on that state to
!       either fail (not prompting) or succeed and issue and error message
!       (prompting).
!

MACRO
        COMMAND_PROMPT (CLS, NAM, STATUS) =

$STATE  (%NAME ('ST_', CLS, ' ', NAM),
        (TPA$_EOS, , ACT$PMT_ON),
        (TPA$_LAMBDA, , ACT$PMT_OFF),
        );

$STATE  (%NAME ('ST_', CLS, '_', NAM, '_1'),

        %REMAINING

        ,
        (TPA$_EOS, %NAME ('ST_', CLS, ' ', NAM, ' 1'),
                ACT$PRMPT, , , %NAME ('PMT$G_', CLS, ' ', NAM) ),
        (TPA$_LAMBDA, %NAME ('ST_', CLS, '_', NAM, '_1'),
                ACT$PMT_Q, , , STATUS)
        );

        %;
```

```
!
!       Build sequence of Query states
!       Query states are states which save a parameter
!       if the answer to a prompt is YES. No parameter is
!       saved for NO or CR.  If the response is "_DONE" then
!       the remainder of the queries are skipped and the function
!       is performed.
!

MACRO
        QUERY_STATES (CLS) [NAM] =

$STATE  (%NAME ('ST_', CLS, '_PMT_', NAM),
        (TPA$_LAMBDA, , ACT$PRMPT, ,
                        %NAME ('PMT$G_', CLS, '_', NAM) )
        );

$STATE  (,
        (TPA$_SYMBOL, %NAME ('ST_', CLS, '_DOIT'), ACT$PMTDONEQ ),
        ( (SE_QRY_YES),
                %IF     %IDENTICAL (NAM, ALL)           ! ALL IS SPECIAL
                %THEN   %NAME ('ST_', CLS, '_DOIT')     ! IT MUST BE LAST
                %FI
                        , ACT$SAVPRM,
                        %NAME ('PBK$G_', CLS, '_', NAM) ),
        ( (SE_QRY_NO) ),
        (TPA$_EOS),
        (TPA$_LAMBDA, %NAME ('ST_', CLS, '_PMT_', NAM),
                        ACT$SIGNAL, , , NCP$_INVVAL)
        );

        %;

!
!       Slightly modified QUERY_STATES macro to permit using
!       same prompt and PBK more than once with multiply defining
!       parse table states.
!

MACRO
        QUERY_STATES_S (CLS) [NAM, SNAM] =

$STATE  (%NAME ('ST_', CLS, '_PMT_', SNAM),
        (TPA$_LAMBDA, , ACT$PRMPT, ,
                        %NAME ('PMT$G_', CLS, '_', SNAM) )
        );

$STATE  (,
        (TPA$_SYMBOL, %NAME ('ST_', CLS, '_DOIT'), ACT$PMTDONEQ ),
        ( (SE_QRY_YES),
                %IF     %IDENTICAL (NAM, ALL)           ! ALL IS SPECIAL
                %THEN   %NAME ('ST_', CLS, '_DOIT')     ! IT MUST BE LAST
                %FI
                        , ACT$SAVPRM,
                        %NAME ('PBK$G_', CLS, '_', NAM) ),
```

```
        ( (SE_QRY_NO) ),
        (TPA$_EOS),
        (TPA$_LAMBDA, %NAME ('ST_', CLS, '_PMT_', SNAM),
                      ACT$SIGNAL, , , NCP$_INVVAL)
        );

        %;

!
!       Build transitions in a dispatch state
!
!   KEY         Keyword for dispatch from state
!

MACRO
    DISPATCH_STATES (CLS) [NAM, KEY] =

        (%STRING (KEY), %NAME ('ST_', CLS, '_PRC_', NAM) )
    %;
```

```
!
!       Build a sequence of process states
!
!   NOISE       Noise keyword
!
MACRO
    PROCESS_STATES (CLS) [NAM, NOISE] =

$STATE  (%NAME ('ST_', CLS, '_PRC_', NAM),
        %IF NOT %NULL (NOISE)
        %THEN
        (%STRING (NOISE)),
        (TPA$_LAMBDA)
        );
$STATE  (,
        %FI
        ( (%NAME ('ST_', CLS, '_', NAM) ),
            %NAME ('ST_', CLS, '_PRC') )
        );
    %;

!
!       Build a set of subexpressions to decode parameters
!
!   TYP         Type of transition desired
!
MACRO
    SUB_EXPRESSIONS (CLS) [NAM, TYP] =

$STATE  (%NAME ('ST_', CLS, '_', NAM),
        (TYP,

        %IF %IDENTICAL (TYP, TPA$_DECIMAL)
        %THEN
            , ACT$NUM_RNG, ,
                        NUM_RANGE
                        (
                        %NAME ('LOW_', CLS, '_', NAM),
                        %NAME ('HIGH_', CLS, '_', NAM)
                        )
                )
        );
$STATE  (,
        (TPA$_LAMBDA,

        %FI

        TPA$_EXIT, ACT$SAVPRM, , ,
            %NAME ('PBK$G_', CLS, '_', NAM) )
        );
    %;
!
```

```
!       Build transitions in a keyword state
!
!       Each transition saves a parameter based on the keyword
!       and exits the subexpression.
!

MACRO
        KEYWORD_STATE (CLS) [NAM, KEY] =

        (%STRING (KEY), TPA$_EXIT, ACT$SAVPRM, , ,
                %NAME ('PBK$G_', CLS, '_', NAM)')
        %;
```

```
%SBTTL  'Macro to Build Prompt Strings'

!
!       Build prompt strings
!

MACRO
    PROMPT_STRINGS (CLS) [NAM, STR] =

%NAME ('PMT$G_', CLS, '_', NAM) =
        ASCID (%STRING (STR) )
    %;
```

```
%SBTTL  'Macros to Build Parameter Control Blocks'

        Build parameter blocks


        There are four structures associated with building messages:

                SDB     Set/Define Block

        This block is a parameter to the verb routines.  It serves
        to point to other structures and to declare the type of the
        entity so that message headers can be properly built.

                PDB     Parameter Data Block

        This is a data area which holds the actual parameter data.
        THe block is a status byte followed by the data as it
        appears in the message.  The action routine ACT$SAVPRM
        stores the data in this block in the correct format.

                PBK     Parameter Block

        This block is a parameter to ACT$SAVPRM and directs the
        storage of the parameter in the PDB.  It contains the type
        of the parameter, the PDB address and an optional parameter
        for the type code.

                PCL     Parameter Control List

        This block is a list of items which control the building of
        messages.  Each entry is a parameter type code, the
        parameter ID code and the PDB address.  Using this block the
        routines which build messages are able to add parameter
        values or codes to the end of messages in the proper format.
```

```
!
!       Build the SDB
!
!
!       CLS     Class of the command
!       ENT     Entity type code.  If negative, then system-specific entity
!       PDB     Parameter data block suffix
!       PCL     PCL suffix
!

MACRO
    BUILD_SDB (CLS, ENT, PDB, PCL) =

!
!   Declare symbols which are not yet declared
!

    %IF NOT %DECLARED (%NAME ('PDB$G_', PDB) )
    %THEN
        EXTERNAL
        %NAME ('PDB$G_', PDB)
        :
    %FI

!
!   Build the PLIT for the SDB
!

    BIND

    %NAME ('SDB$G_', CLS) =

    UPLIT BYTE                                  ! Use byte alignment to
        (                                       ! Save space
        BYTE (ENT),
        LONG (%NAME ('PDB$G_', PDB) ),
        LONG (%NAME ('PCL$G_', PCL) )
        )
    :
    %;
```

```
!
!       Build a PCL
!
!
!       CLS     Class of command
!
!       remaining repeated
!
!       NAM     Name of parameter concerned
!       TYP     Suffix for type code
!       ID      Suffix for parameter ID code
!       PDB     Suffix for PDB of data
!
MACRO
    BUILD_PCL (CLS) =
!
!   Declare the PDB's
!
    BUILD_PCL_PDB (CLS, %REMAINING)
!
!   Build the PCL PLIT
!
    BIND
    %NAME ('PCL$G_',CLS) =
    UPLIT BYTE                          ! Use byte alignment to save space
        (
        BUILD_PCL_LST (CLS, %REMAINING)
        )
    %,
!
!   Build the items in the PCL list
!
    BUILD_PCL_LST (CLS) [NAM, TYP, ID, PDB] =
    BYTE (%NAME ('PBK$K_', TYP) ),      ! Data type code
    WORD (
        %IF %NULL (ID)                  ! Network management ID
        %THEN 0
        %ELSE %NAME ('NMA$C_',ID)
        %FI
        ),
    LONG (                              ! Address of PDB
        %IF %NULL (NAM)
        %THEN 0
```

```
        %ELSE %NAME ('PDB$G_',
                     %IF %NULL (PDB)
                     %THEN CLS, '_', NAM
                     %ELSE PDB
                     %FI
                     )
        %FI
        )

    %,

!
!   Declare the PDB as external
!

    BUILD_PCL_PDB (CLS) [NAM, I2, I3, PDB] =

    %IF NOT %NULL (NAM)
    %THEN
        %IF NOT %DECLARED
              (%NAME ('PDB$G_',
                     %IF %NULL (PDB)
                     %THEN CLS, '_', NAM
                     %ELSE PDB
                     %FI
                     )
               )
        %THEN

            EXTERNAL
                  %NAME ('PDB$G_',
                         %IF %NULL (PDB)
                         %THEN CLS, '_', NAM
                         %ELSE PDB
                         %FI
                         )
                :
        %FI
    %FI
    %;
```

```
!
!       Build a list of PBK's
!
!
!       CLS     Class of command
!
!       remaining are repeated
!
!       NAM     Suffix name of parameter
!       TYP     Suffix of type code of parameter
!       PRM     Value of type code parameter
!       PDB     Suffix for PDB to save parameter
!

MACRO
    BUILD_PBK (CLS) [NAM, TYP, PRM, PDB] =

    %IF NOT %DECLARED                           ! Declare the pdb external
        (%NAME ('PDB$G_',
                    %IF %NULL (PDB)
                    %THEN CLS, '_', NAM
                    %ELSE PDB
                    %FI
                )
        )
    %THEN

        EXTERNAL
            %NAME ('PDB$G_',
                        %IF %NULL (PDB)
                        %THEN CLS, '_', NAM
                        %ELSE PDB
                        %FI
                    )
            ;

    %FI

    GLOBAL BIND                             ! Build PBK as a plit

    %NAME ('PBK$G_', CLS, '_', NAM) =

    UPLIT BYTE                              ! Use byte alignment to save space
        (
        BYTE (%NAME ('PBK$K_', TYP) ),      ! Data type code
        LONG (%NAME ('PDB$G_',              ! PDB address
                        %IF %NULL (PDB)
                        %THEN CLS, '_', NAM
                        %ELSE PDB
                        %FI
                    )
            ),
        LONG (                              ! Parameter for type code routine
```

```
            %IF %NULL (PRM)
            %THEN 0
            %ELSE PRM
            %FI
            )
        )

    %;
```

```
!
!       Build a PDB
!

!
!       CLS     Class of command
!       NAM     Suffix for parameter
!       SIZ     Size of parameter data in bytes
!

MACRO
    BUILD_PDB (CLS) [NAM, SIZ] =

    %NAME ('PDB$G_', CLS, '_', NAM) :           ! Name in classic form
            BLOCK [(SIZ) + 1, 1]                ! Size + 1 for status byte
            ALIGN (0)                           ! Byte align to save space

    %;
```

```
!
!       Build a numeric range parameter
!

MACRO
    NUM_RANGE (LOW, HIGH) =

        ( UPLIT BYTE ( LONG ( (LOW), (HIGH) ) ) ) ! Byte align to save space
        %;

!
!       Build a next state parameter
!

MACRO
        NEXT_STATE (COD) =
        (UPLIT BYTE                     ! Byte align to save space
                (
                LONG                    ! Each is an address in a longword
                    (
                    %NAME ('NCP$G_STTBL_',COD),
                    %NAME ('NCP$G_KYTBL_', COD)
                    )
                )
        )
        %;
```

```
%SBTTL  'Equated Symbols'

!
! EQUATED SYMBOLS:
!

LITERAL
        TRUE            = 1,
        FALSE           = 0,
        SUCCESS         = 1,
        FAILURE         = 0,

        NCP$C_VRS       = 4,    ! Version of NCP for messages
        NCP$C_ECO       = 0,    ! Eco for messages
        NCP$C_UECO      = 0,    ! User eco for messages

        NCP$C_MBXSIZ = 40,      ! Size of the mailbox buffer for network io
        NCP$C_RSPSIZ = 1000,    ! Size of the response buffer for network io

        LEN_OBJ_NAM     = 12,   ! Length of an object name
        LEN_ID_STR      = 32,   ! Length of an ID string
        LEN_NSP_PSW     = 8,    ! Length of a nsp password
        LEN_FILE_SPEC   = 64,   ! Length of a file spec
        LEN_FILE_NAM    = 9,    ! Length of a file name
        LEN_FILE_TYP    = 3,    ! Length of a file type
        LOW_NODE_ADR    = 0,    ! Low limit of node address
        HIGH_NODE_ADR   = 1023, ! High limit
        LEN_NODE_NAM    = 6,    ! Length of node name
        LEN_NI_ADR      = 6,    ! Length of NI Address
        LOW_AREA        = 1,    ! Low limit of a node area
        HIGH_AREA       = 65,   ! High limit
        LEN_CIRC_ID     = 16,   ! Length of a circuit id
        LEN_LINE_ID     = 16,   ! Length of a total line id
        LEN_HEX_NUM     = 32,   ! Length of Hex number (128 bits)
        LEN_HEX_PSW     = 16,   ! Length of Hex password (64 bits)
        LEN_ACC_ACC     = 39,   ! Length of the access account
        LEN_ACC_PSW     = 39,   ! Length of the access password
        LEN_ACC_USR     = 39,   ! Length of the access user id
        LOW_EVENT_CLS   = 0,    ! Low limit of event class
        HIGH_EVENT_CLS  = 511,  ! High limit
        LOW_EVENT_TYP   = 0,    ! Low limit of event type
        HIGH_EVENT_TYP  = 31,   ! High limit
        LEN_PRV_MSK     = 8,    ! Length in bytes of a priv mask
        LEN_SOFT_ID     = 16,   ! Length of a node software id
        LOW_UIC_PART    = 0,    ! Low limit of uic number
        HIGH_UIC_PART   = 255,  ! High limit
        LEN_DTE_NUM     = 16,   ! Length of X.25 circuit DTE address
        LEN_ENT_NAM     = 16,   ! Length of entity name
        LEN_GRP_NAME    = 16,   ! Length of X.25 closed user group name
        LEN_NET_NAME    = 16,   ! Length of X.25 network name
        LEN_DEST_NAME   = 16,   ! Length of X.25 destination name
        LEN_TRCPNT_NAME = 31,   ! Length of X.25 tracepoint name
        MAX_RNGLST_PAIRS= 16;   ! Maximum numbers of pairs in a range list
```

```
!
!       Macro to help define ranges
!

MACRO
        DEFRNG (CLS) [NAM, LO, HI] =

        LITERAL
                %NAME ('HIGH_', CLS, '_', NAM) = HI,
                %NAME ('LOW_', CLS, '_', NAM) = LO
                ;
        %;

DEFRNG (NOD,                    ! Executor node parameters

        ADR, 0, 1023,           ! Node address
        AMC, 1, 65535,          ! Area maximum cost
        AMH, 1, 255,            ! Area maximum hops
        BRT, 1, 65535,          ! Broadcast routing timer
        BSZ, 1, 65535,          ! Buffer size
        DFC, 1, 255,            ! Delay factor
        DWT, 1, 255,            ! Delay weight
        FBS, 1, 65535,          ! Forwarding buffer size
        IAT, 1, 65535,          ! Inactivity timer
        INT, 1, 65535,          ! Incoming timer
        MAD, 1, 65535,          ! Max address
        MAR, 1, 255,            ! Max area
        MBE, 1, 65535,          ! Max broadcast nonrouters
        MBR, 1, 65535,          ! Max broadcast routers
        MBF, 0, 65535,          ! Max buffers
        MCO, 1, 1023,           ! Max cost
        MHP, 1, 31,             ! Max hops
        MLN, 1, 65535,          ! Max lines
        MLK, 1, 65535,          ! Max links
        MVS, 1, 255,            ! Max visits
        OTM, 1, 65535,          ! Outgoing timer
        RFC, 1, 65535,          ! Retransmit factor
        RTM, 1, 65535,          ! Routing timer
        SBS, 1, 65535,          ! Segment buffer size
        PIQ, 0, 65535)          ! Pipeline quota

DEFRNG (CIR,                    ! Circuit parameters

        CTM, 1, 65535,          ! Counter timer
        COS, 1, 25,             ! Cost
        MRT, 0, 255,            ! Maximum routers on NI
        RPR, 0, 127,            ! Router priority on NI
        HET, 1, 65535,          ! Hello timer
        LIT, 1, 65535,          ! Listen timer
        MRC, 0, 255,            ! Maximum recalls
        RCT, 1, 65535,          ! Recall timer
        CHN, 0, 4095,           ! Channel number
        MBL, 1, 65535,          ! Maximum block
```

```
        MWI, 1, 255,            ! Maximum window
        TRI, 0, 255,            ! Tributary address
        BBT, 1, 65535,          ! Babble timer
        TRT, 0, 65535,          ! Transmit timer
        MTR, 1, 255,            ! Maximum transmits
        ACB, 0, 255,            ! Active base
        ACI, 0, 255,            ! Active increment
        IAB, 0, 255,            ! Inactive base
        IAI, 0, 255,            ! Inactive increment
        IAT, 0, 255,            ! Inactive threshold
        DYB, 0, 255,            ! Dying base
        DYI, 0, 255,            ! Dying increment
        DYT, 0, 255,            ! Dying threshold
        DTH, 0, 255)            ! Dead threshold

DEFRNG (LIN,                    ! Line parameters

        CTM, 1, 65535,          ! Counter timer
        BLO, 0, 65535,          ! Block size
        COS, 1, 25              ! Cost of the line
        NTM, 1, 65535,          ! Normal timer
        STM, 1, 65535,          ! Service timer
        RTT, 1, 65535,          ! Retransmit timer
        HTI, 1, 65535,          ! Holdback timer
        MBL, 1, 65535,          ! Maximum block
        MRT, 1, 255,            ! Maximum retransmits
        MWI, 1, 255,            ! Maximum window
        TRB, 0, 255,            ! Tributary address
        SLT, 50, 65535,         ! Scheduling timer
        DDT, 1, 65535,          ! Dead timer
        DLT, 1, 65535,          ! Delay timer
        SRT, 0, 65535,          ! Stream timer
        BFN, 1, 1024,           ! Number of buffers
        BFS, 1, 65535)          ! Buffer size

DEFRNG (LOO,                    ! Loop parameters

        CNT, 1, 65535,          ! Count of messages
        LEN, 1, 65535)          ! Length of message in bytes

DEFRNG (LNK,                    ! Link parameter

        ADR, 1, 65535)          ! Link address

DEFRNG (NOD,                    ! Node parameters

        CTM, 1, 65535,          ! Counter timer
        DCT, 0, %X'FFFFFFFF')   ! Dump count

DEFRNG (DUM,
```

```
        COU, 0, %X'FFFFFFFF')   ! Dump count

DEFRNG (OBJ,                    ! Object parameters
        NUM, 0, 255)            ! Object number
DEFRNG (MCS,                    ! Module Console
        RTR, 0, 65535)          ! Object number
DEFRNG (MPR,                    ! X25-PROTOCOL
        CTM, 1, 65535,          ! Counter timer
        DBL, 1, 65535,          ! Default block
        DWI, 1, 127,            ! Default window
        MBL, 16, 4096,          ! Maximum block
        MWI, 1, 127,            ! Maximum window
        MCL, 1, 255,            ! Maximum clears
        MRS, 1, 255,            ! Maximum resets
        MST, 1, 255,            ! Maximum restarts
        CAT, 1, 255,            ! Call timer
        CLT, 1, 255,            ! Clear timer
        RST, 1, 255,            ! Reset timer
        STT, 1, 255,            ! Restart timer
        GNM, 0, 9999,           ! Closed user group number
        MCI, 1, 65535           ! Maximum circuits - VMS specific
        )
DEFRNG (MSE,                    ! X25-SERVER
        CTM, 1, 65535,          ! Counter timer
        MCI, 1, 65535,          ! Maximum circuits
        PRI, 0, 255)            ! Priority
DEFRNG (MTR,                    ! X25-TRACE
        BSZ, 1, 4096,           ! Buffer size
        CPL, 1, 65535,          ! Capture limit
        CPS, 1, 65535,          ! Capture size
        MBK, 1, 65535,          ! Maximum blocks
        MBF, 1, 65535,          ! Maximum buffers
        MVR, 1, 63)             ! Maximum versions
```

```
%SBTTL  'Macro to Define External Symbols'

!
! EXTERNAL REFERENCES:
!

!
!       Define externals for action routines
!

MACRO
        ACT_DFN =

EXTERNAL ROUTINE
        ACT$INV_COMMAND,                ! Signal invalid command
        ACT$SAVPRM,                     ! Save a parameter
        ACT$TMPSTR,                     ! Save a temporary string
        ACT$BLNK_SIG,                   ! Blanks are now significant
        ACT$BLNK_NSIG,                  ! Blanks are not significant
        ACT$ZAPTMPDSC,                  ! Clear temporary descriptors
        ACT$PRMPT,                      ! Prompt for a parameter
        ACT$NUM_RNG,                    ! Validate a number
        ACT$NUM_RNGSAV,                 ! Validate and store range list number
        ACT$NUM_SAV,                    ! Store a number from a range list
        ACT$STR_LEN,                    ! Validate a string length
        ACT$WRI_STR,                    ! Write a string to SYS$OUTPUT
        ACT$SIGNAL,                     ! Signal an error condition
        ACT$PMT_ON,                     ! Prompting on
        ACT$PMT_OFF,                    ! Prompting off
        ACT$PMT_Q,                      ! Check prompting
        ACT$VRB_LOOP,                   ! Loop Verb processing
        ACT$VRB_UTILITY,                ! Most other Verbs
        ACT$VRB_SHOLIS,                 ! Show and List Verbs
        ACT$CLRLONG,                    ! Clear a longword
        ACT$TESTLONG,                   ! Test a longword
        ACT$COPY_VALUE,                 ! Copy a longword
        ACT$PMTDONEQ                    ! See if prompting done
        ;

EXTERNAL
        PBK$G_ZAPACCDSC,                ! Parameter block to zap descriptors

        PBK$G_VRB_ALL,                  ! Block for All parameter
        PBK$G_LOG_TYPCON,               ! Block for logging types
        PBK$G_LOG_TYPFIL,
        PBK$G_LOG_TYPMON,

        PBK$G_EVE_ESET,                 ! Parameter blocks for events
        PBK$G_EVE_ECLS,
        PBK$G_EVE_EMSK,
        PBK$G_EVE_ERNG,
        PBK$G_EVE_EWLD,
        PBK$G_EVE_ESNO,
        PBK$G_EVE_ESLI,
        PBK$G_EVE_ESEX,
```

```
        NCP$GL_OPTION,                  ! Place to build option byte
        NCP$GL_FNC_CODE                 ! Place to build function code

        ;
```

```
!
!       String descriptors for access parameters
!
EXTERNAL
        ACT$GL_ADR_Q,                   ! Flag for address
        ACT$GL_NODAREA,                 ! Node Area
        ACT$GQ_ACCACC_DSC,              ! Account
        ACT$GQ_ACCPSW_DSC,              ! Password
        ACT$GQ_ACCUSR_DSC,              ! User id
        ACT$GQ_NODEID_DSC,              ! Node id descriptor
        ACT$GL_SAD_BEGIN,               ! Subaddress beginning value
        ACT$GL_SAD_END;                 ! Subaddress ending value
!
!       Status return values
!
EXTERNAL LITERAL
        NCP$_INVVAL,                    ! Unrecognised value
        NCP$_INVKEY                     ! Unrecognised keyword
        ;               !

        %;
```

%SBTTL 'Macros to Build Subexpressions'

The state tables for the NCP language have been broken into smaller modules to reduce . pile time of the separate modules to reduce development time. The developement time has been reduced at the expense of a slight increase in the size of the tables since keywords and subexpression states are duplicated in the separate tables.

These macros define whole state subexpressions to parse useful entities. Including these subexpressions as macros in the library avoids having multiple copies of the source of the subexpressions in each of the modules of the states tables where they are used.

States and subexpressions are named in a distinctive way. States are named ST_xxx. Subexpressions are named SE_xxx and subexpression defining macros are named SEM_xxx.

```
!
!       Subexpression for a File ID
!
MACRO   SEM_FILE_ID =

$STATE  (SE_FILE_ID,                    ! Make blanks significant
        (TPA$_EOS, TPA$_FAIL),
        (TPA$_LAMBDA,   , ACT$BLNK_SIG));

!
!       Accept any string of characters for a filespec.  Format is not
!       enforced here.
!

$STATE  (SE_FILE_ID1,
        (TPA$_EOS, SE_FILE_IDX),
        (TPA$_BLANK, SE_FILE_IDX),
        ('"', SE_FILE_ID2),             ! Handle quoted portion separately
        (TPA$_ANY, SE_FILE_ID1));

$STATE  (SE_FILE_ID2,
        ('"', SE_FILE_ID1),             ! If ending double quote, rejoin loop
        (TPA$_EOS, SE_FILE_IDE),
        (TPA$_ANY, SE_FILE_ID2));

$STATE  (SE_FILE_IDX,
        (TPA$_LAMBDA,   TPA$_EXIT, ACT$BLNK_NSIG));

$STATE  (SE_FILE_IDE,
        (TPA$_LAMBDA,   TPA$_FAIL, ACT$BLNK_NSIG));

%;                      ! End of File-id macro
```

```
!
!       Subexpression for Node-ID
!

MACRO   SEM_NODE_ID =

$STATE  (SE_NODE_ID,
        ( (SE_NODE_NAM), TPA$_EXIT),
        ( (SE_NODE_ADR), TPA$_EXIT)
        );

$STATE  (SE_NODE_ADR,
        ( (SE_NOD_ADR), TPA$_EXIT, , TRUE, ACT$GL_ADR_Q)
        );

$STATE  (SE_NOD_ADR,
        (TPA$_LAMBDA, , ACT$CLRLONG, , , ACT$GL_ADR_Q)
        );
$STATE  (,
        ((SE_NODE_AREA_Q), TPA$_EXIT),                          ! If an area precedes the adr then check its range and store it
        (TPA$_DECIMAL, TPA$_EXIT, ACT$NUM_RNG, , ,
                NUM_RANGE (LOW_NODE_ADR, HIGH_NODE_ADR))
        );


$STATE  (SE_NODE_NAM,
        (TPA$_LAMBDA,    , ACT$BLNK_SIG)
        );

$STATE  (,
        (TPA$_LAMBDA, , ACT$CLRLONG, , , ACT$GL_ADR_Q)
        );

$STATE  (,
        ( (SE_NODE_NAM1), , ACT$STR_LEN, , , LEN_NODE_NAM),
        (TPA$_LAMBDA,   TPA$_FAIL, ACT$BLNK_NSIG)
        );

$STATE  (,
        (TPA$_LAMBDA,   TPA$_EXIT, ACT$BLNK_NSIG)
        );

$STATE  (SE_NODE_NAM1,
        (TPA$_DIGIT,    SE_NODE_NAM1),                          ! Check for Node names with leading digits
        (TPA$_ALPHA),                                           ! If the node name has an alpha then drop to ST_NODE_NAM2
        ('$')                                                   !  Otherwise it was only digits and therefore an ADR, so fail.
        );

$STATE  (ST_NODE_NAM2,
        (TPA$_DIGIT,    ST_NODE_NAM2),
        (TPA$_ALPHA,    ST_NODE_NAM2),
        ('$'            ST_NODE_NAM2),
        (TPA$_LAMBDA,   TPA$_EXIT)
        );
```

```
!
!       See if the node address has an area in front.
!       Format is area.adr, where area and adr are decimal.
!
$STATE  (SE_NODE_AREA_Q,
        (TPA$_DECIMAL, , , ,ACT$GL_NODAREA)                     ! Store it in case it was an area
        );

$STATE  (,
        ('.', , ACT$NUM_RNG, , ,                                ! The last number parsed was indeed an area
                NUM_RANGE (LOW_AREA, HIGH_AREA)),               !  so check the range
        (TPA$_LAMBDA, TPA$_FAIL, ACT$CLRLONG,,, ACT$GL_NODAREA) ! There was no area so clear storage
        );

$STATE  (,
        (TPA$_DECIMAL, TPA$_EXIT, ACT$NUM_RNG, , ,              ! Check the range of the node address
                NUM_RANGE (LOW_NODE_ADR, HIGH_NODE_ADR))
        );

        %;
```

```
!
!       Check range of node area number
!

MACRO   SEM_AREA_NUM =

$STATE  (SE_AREA_NUM,
        (TPA$_DECIMAL, TPA$_EXIT, ACT$NUM_RNG, , ,
                NUM_RANGE (LOW_AREA, HIGH_AREA)),
        );
        %;
```

```
!
!       Subexpression to accept NI address of the form nn-nn-nn-nn-nn-nn
!       and since we're really nice, as a bonus we'll take nnnnnnnnnnnn.
!

MACRO   SEM_NI_ADR =

$STATE  (SE_NI_ADR,
        ((SE_NI_ADDR), TPA$_EXIT),
        ((SE_NI_NUM), TPA$_EXIT)
        );

!
!       Only accepts nn-nn-nn-nn-nn-nn
!
$STATE  (SE_NI_ADDR,
        (TPA$_LAMBDA, , ACT$BLNK_SIG)
        );

$STATE  (,
        ((SE_NUM_PAIR)));
$STATE  (,
        ('-')
        );

$STATE  (,
        ((SE_NUM_PAIR)));
$STATE  (,
        ('-')
        );

$STATE  (,
        ((SE_NUM_PAIR)));
$STATE  (,
        ('-')
        );

$STATE  (,
        ((SE_NUM_PAIR)));
$STATE  (,
        ('-')
        );

$STATE  (,
        ((SE_NUM_PAIR)));
$STATE  (,
        ('-')
        );

$STATE  (,
        ((SE_NUM_PAIR), TPA$_EXIT, ACT$BLNK_NSIG),
        (TPA$_LAMBDA, TPA$_FAIL, ACT$BLNK_NSIG)
        );

!
!       Accept two Hex digits
```

```
!
$STATE  (SE_NUM_PAIR,
        ((SE_HEX_DIGIT)),
        (TPA$_LAMBDA, SE_PAIR_FAIL));

$STATE  (,                          ! 2nd Hex digit
        ((SE_HEX_DIGIT), TPA$_EXIT),
        (TPA$_LAMBDA, SE_PAIR_FAIL));

$STATE  (SE_PAIR_FAIL,
        (TPA$_LAMBDA, TPA$_FAIL, ACT$BLNK_NSIG));
```

```
!
!       Accept a twelve digit hex number
!
$STATE  (SE_NI_NUM,
        ((SE_HEX_DIGIT)) );
$STATE  (,
        ((SE_HEX_DIGIT)) );
$STATE  (,
        ((SE_HEX_DIGIT)) );
$STATE  (,
        ((SE_HEX_DIGIT)) );
$STATE  (,
        ((SE_HEX_DIGIT)) );
$STATE  (,
        ((SE_HEX_DIGIT)) );
$STATE  (,
        ((SE_HEX_DIGIT)) );
$STATE  (,
        ((SE_HEX_DIGIT)) );
$STATE  (,
        ((SE_HEX_DIGIT)) );
$STATE  (,
        ((SE_HEX_DIGIT)) );
$STATE  (,                      ! 12th hex digit
        ((SE_HEX_DIGIT), TPA$_EXIT, ACT$BLNK_NSIG)
        );

$STATE  (SE_HEX_DIGIT,          ! Accept valid hex digit
        (TPA$_DIGIT, TPA$_EXIT),
        ('A', TPA$_EXIT),
        ('B', TPA$_EXIT),
        ('C', TPA$_EXIT),
        ('D', TPA$_EXIT),
        ('E', TPA$_EXIT),
        ('F', TPA$_EXIT));

        %;
```

```
        Subexpression for Link ID

        (This subexpression restricts the link ID to be a number within
        the range of 0-65535.  However, the NADR entity is used to store
        the link ID because the format is similiar: format byte of zero,
        followed by the word link address.  The format byte is used to
        enable requests of known links; format byte of -1).

MACRO   SEM_LINK_ID =

$STATE  (SE_LINK_ID,
        (TPA$_LAMBDA,, ACT$CLRLONG,,, ACT$GL_ADR_Q));

$STATE  (,
        (TPA$_DECIMAL, TPA$_EXIT, ACT$NUM_RNG, TRUE, ACT$GL_ADR_Q,
                        NUM_RANGE(0, 65535)));

        %;
```

```
!
!       Hex Password for Service Operations
!

MACRO   SEM_HEX_PSW =

$STATE  (SE_HEX_PSW,
        ( (SE_HEX_STR), TPA$_EXIT, ACT$STR_LEN, , , LEN_HEX_PSW));

$STATE  (SE_HEX_STR,
        (TPA$_LAMBDA, , ACT$BLNK_SIG));

$STATE  (,
        ((SE_HEX_CHR)),         ! Ensure at least one character given
        (TPA$_LAMBDA, TPA$_FAIL, ACT$BLNK_NSIG));

$STATE  (SE_HEX_STR1,
        ((SE_HEX_CHR), SE_HEX_STR1),    ! Gobble remaining hex characters
        ((SE_HEX_NONTERM), TPA$_FAIL, ACT$BLNK_NSIG),
        (TPA$_LAMBDA,   TPA$_EXIT, ACT$BLNK_NSIG));

$STATE  (SE_HEX_NONTERM,        ! Return false if terminator, else true
        (TPA$_BLANK,TPA$_FAIL), ! (so that blank is not gobbled by TPARSE)
        (TPA$_EOS,TPA$_FAIL),
        (TPA$_LAMBDA,TPA$_EXIT));

$STATE  (SE_HEX_CHR,            ! True if valid hex char (gobbled), else false
        (TPA$_DIGIT, TPA$_EXIT),
        ('A', TPA$_EXIT),
        ('B', TPA$_EXIT),
        ('C', TPA$_EXIT),
        ('D', TPA$_EXIT),
        ('E', TPA$_EXIT),
        ('F', TPA$_EXIT));

%;
```

```
!
!       Hex Number
!

MACRO   SEM_HEX_NUM =

$STATE  (SE_HEX_NUM,
        ( (SE_HEX_STR), TPA$_EXIT, ACT$STR_LEN, , , LEN_HEX_NUM));

$STATE  (SE_HEX_STR,
        (TPA$_LAMBDA, , ACT$BLNK_SIG));

$STATE  (,
        ((SE_HEX_CHR)),         ! Ensure at least one character given
        (TPA$_LAMBDA, TPA$_FAIL, ACT$BLNK_NSIG));

$STATE  (SE_HEX_STR1,
        ((SE_HEX_CHR), SE_HEX_STR1),    ! Gobble remaining hex characters
        ((SE_HEX_NONTERM), TPA$_FAIL, ACT$BLNK_NSIG),
        (TPA$_LAMBDA,   TPA$_EXIT, ACT$BLNK_NSIG));

$STATE  (SE_HEX_NONTERM,        ! Return false if terminator, else true
        (TPA$_BLANK,TPA$_FAIL), ! (so that blank is not gobbled by TPARSE)
        (TPA$_EOS,TPA$_FAIL),
        (TPA$_LAMBDA,TPA$_EXIT));

$STATE  (SE_HEX_CHR,            ! True if valid hex char (gobbled), else false
        (TPA$_DIGIT, TPA$_EXIT),
        ('A', TPA$_EXIT),
        ('B', TPA$_EXIT),
        ('C', TPA$_EXIT),
        ('D', TPA$_EXIT),
        ('E', TPA$_EXIT),
        ('F', TPA$_EXIT));

%;
```

```
!
!       Subexpression for a circuit name
!

MACRO
        SEM_CIRC_ID =

$STATE  (SE_CIRC_ID,
        ((SE_LINE), TPA$_EXIT, ACT$STR_LEN, , , LEN_CIRC_ID)
        );

        %;

!
!       Subexpression for a DTE call number
!

MACRO   SEM_DTE_NUMBER =

$STATE  (SE_DTE_NUMBER,
        (TPA$_STRING, TPA$_EXIT, ACT$STR_LEN,,, LEN_DTE_NUM)
        );

        %;

!
!       Subexpression for a closed user group name
!

MACRO   SEM_GRP_NAME =

$STATE  (SE_GRP_NAME,
        (TPA$_SYMBOL, TPA$_EXIT, ACT$STR_LEN,,, LEN_GRP_NAME)
        );

        %;

!
!       Subexpression for an X.25 network name
!

MACRO   SEM_NET_NAME =

$STATE  (SE_NET_NAME,
        (TPA$_SYMBOL, TPA$_EXIT, ACT$STR_LEN,,, LEN_NET_NAME)
        );

        %;

!
!       Subexpression for an X.25 destination name
!

MACRO   SEM_DEST_NAME =

$STATE  (SE_DEST_NAME,
```

```
        (TPA$_SYMBOL, TPA$_EXIT, ACT$STR_LEN,,, LEN_DEST_NAME)
        );

        %;
```

```
        Subexpression for a subaddress range of the form:

        number
        number-number

MACRO   SEM_SUBADR_RANGE =

$STATE  (SE_SUBADR_RANGE,
        (TPA$_DECIMAL,, ACT$NUM_RNG,, ACT$GL_SAD_BEGIN,
                NUM_RANGE (0, 9999)));

$STATE  (,
        (TPA$_LAMBDA,, ACT$COPY_VALUE,,, ACT$GL_SAD_END));

$STATE  (,
        ('-'),
        (TPA$_LAMBDA, TPA$_EXIT));

$STATE  (,
        (TPA$_DECIMAL,TPA$_EXIT, ACT$NUM_RNG,, ACT$GL_SAD_END,
                NUM_RANGE (0, 9999)));

        %;

        Subexpression for a channels list range of the form:

        number
        number, number
        number-number
        number[-number[,..., number[-number]]]

        NOTE: values in channels lists have limit of 4095

MACRO   SEM_RNG_LIST =

$STATE  (SE_RNG_LIST,
        (TPA$_DECIMAL,, ACT$NUM_RNGSAV,,, NUM_RANGE (0, 4095))
        );

$STATE  (,
        (',', SE_RNG_LIST, ACT$NUM_SAV),
        ('-', SE_RNG_HYPHEN),
        (TPA$_LAMBDA, TPA$_EXIT));

$STATE  (SE_RNG_HYPHEN,
        (TPA$_DECIMAL,, ACT$NUM_RNGSAV,,, NUM_RANGE (0, 4095)),
        (TPA$_LAMBDA, TPA$_EXIT));

$STATE  (,
        (',', SE_RNG_LIST),
        (TPA$_LAMBDA, TPA$_EXIT)
        );
```

```
        %;
```

```
!
!       Subexpression for a tracepoint name
!

MACRO
        SEM_TRCPNT_NAME =

$STATE  (SE_TRCPNT_NAME,
        ((SE_FILE_ID), TPA$_EXIT, ACT$STR_LEN, , , LEN_TRCPNT_NAME)
        );

        %;
```

```
!
!       Subexpression for a line ID
!
!       Allow any string terminated with a blank
!

MACRO   SEM_LINE_ID =

$STATE  (SE_LINE_ID,
        ( (SE_LINE),    TPA$_EXIT, ACT$STR_LEN, , , LEN_LINE_ID)
        );

$STATE  (SE_LINE,
        (TPA$_LAMBDA, , ACT$BLNK_SIG)
        );

$STATE  (,
        (TPA$_ALPHA),
        (TPA$_DIGIT),
        ('-'),
        ('.'),
        ('*'),
        ('$')
        );

$STATE  (SE_LINECHAR,
        (TPA$_ALPHA, SE_LINECHAR),
        (TPA$_DIGIT, SE_LINECHAR),
        ('-', SE_LINECHAR),
        ('.', SE_LINECHAR),
        ('*', SE_LINECHAR),
        ('$', SE_LINECHAR),
        (TPA$_LAMBDA, TPA$_EXIT, ACT$BLNK_NSIG)
        );

        %;
```

```
!
!       Subexpression for the ALL parameter
!

MACRO
        SEM_ALL =

$STATE  (SE_ALL,
        ('ALL')         ! If the word is here it must be last on the line
        );

$STATE  (
        (IPA$_EOS, IPA$_EXIT, ACT$SAVPRM, , , PBK$G_VRB_ALL)
        );

        %;
```

```
!
!       Subexpression for Access Control Information
!

MACRO   SEM_ACCESS =

$STATE  (SE_ACC_ACC,
        ( (SE_QOOT_STR), TPA$_EXIT, ACT$STR_LEN, , , LEN_ACC_ACC)
        );

$STATE  (SE_ACC_PSW,
        ( (SE_QOOT_STR), TPA$_EXIT, ACT$STR_LEN, , , LEN_ACC_PSW)
        );

$STATE  (SE_ACC_USR,
        ( (SE_QOOT_STR), TPA$_EXIT, ACT$STR_LEN, , , LEN_ACC_USR)
        );

        %;
```

```
!
!       Subexpression for a quoted string
!

MACRO   SEM_QUOT_STR =

$STATE  (SE_QUOT_STR,
        (TPA$_EOS,      TPA$_FAIL),             ! Got to be something
        (TPA$_BLANK,    ,       ACT$BLNK_SIG),  ! Make blanks significant
        (TPA$_LAMBDA,   ,       ACT$BLNK_SIG)
        );

$STATE  (,                                      ! Quoted string or just string
        ('"',           ST_QUOT_STR3),
        (TPA$_LAMBDA)
        );

$STATE  (ST_QUOT_STR2,                          ! Just a string
        (TPA$_SYMBOL,   ST_QUOT_STR2),
        (TPA$_BLANK,    ST_QUOT_STRX),
        (TPA$_ANY,      ST_QUOT_STR2),
        (TPA$_EOS,      ST_QUOT_STRX)
        );

$STATE  (ST_QUOT_STR3,                          ! A quoted string to be sure
        ( (SE_QUOT_DBL), ST_QUOT_STR3),
        ('"',           ST_QUOT_STRX),
        (TPA$_ANY,      ST_QUOT_STR3),
        (TPA$_EOS,      ST_QUOT_STRE)
        );

$STATE  (ST_QUOT_STRX,
        (TPA$_LAMBDA,   TPA$_EXIT, ACT$BLNK_NSIG)
        );

$STATE  (ST_QUOT_STRE,
        (TPA$_LAMBDA,   TPA$_FAIL, ACT$BLNK_NSIG)
        );

$STATE  (SE_QUOT_DBL,                   ! Do we have a double quote
        ('"')
        );

$STATE  (,
        ('"',   TPA$_EXIT)
        );

        %;
```

```
!
!       Event list subexpression
!

MACRO
        SEM_EVENT_LIST =

$STATE  (SE_EVENT_LIST,
        (TPA$_LAMBDA, , ACT$BLNK_SIG)
        );

$STATE  (,
        ( (SE_EVENT), TPA$_EXIT, ACT$BLNK_NSIG),
        (TPA$_LAMBDA, TPA$_FAIL, ACT$BLNK_NSIG)
        );


!
!       Parse a single event
!

$STATE  (SE_EVENT,
        (TPA$_DECIMAL, ,ACT$NUM_RNG, , ,
                        NUM_RANGE (LOW_EVENT_CLS, HIGH_EVENT_CLS) ),
        );

$STATE  (,
        (TPA$_LAMBDA, , ACT$SAVPRM, , , PBK$G_EVE_ECLS)
        );

$STATE  (,
        ('.')
        );

$STATE  (ST_EVENT_1,
        ( (SE_EVENT_TYP), , ACT$SAVPRM , , , PBK$G_EVE_EMSK),
        ('*', TPA$_EXIT, ACT$SAVPRM, 2^(14+8), PDB$G_VRB_EVE, PBK$G_EVE_EWLD)
        );

$STATE  (,
        ('.', ST_EVENT_1),
        ('-', ST_EVENT_2),
        (TPA$_BLANK, TPA$_EXIT),
        (TPA$_EOS, TPA$_EXIT)
        );
```

```
$STATE  (ST_EVENT_2,
        ( (SE_EVENT_TYP), , ACT$SAVPRM, , , PBK$G_EVE_ERNG)
        );

$STATE  (,
        (',', ST_EVENT_1),
        (TPA$_BLANK, TPA$_EXIT),
        (TPA$_EOS, TPA$_EXIT)
        );

!
!       Known events
!

$STATE  (SE_EVENT_KNOWN,
        (TPA$_LAMBDA, TPA$_EXIT, ACT$SAVPRM, 3^(14+8), PDB$G_VRB_EVE,
                                PBK$G_EVE_EWLD)
        );


!
!       Parse the type for an event
!

$STATE  (SE_EVENT_TYP,
        (TPA$_DECIMAL, TPA$_EXIT, ACT$NUM_RNG, , ,
                        NUM_RANGE (LOW_EVENT_TYP, HIGH_EVENT_TYP) )
        );

        %;
```

```
!
!       Logging type
!

MACRO
        SEM_LOG_TYP =

$STATE  (SE_LOG_TYP,

        KEYWORD_STATE
        (LOG,

        TYPCON, 'CONSOLE',
        TYPFIL, 'FILE',
        TYPMON, 'MONITOR',

        )
        );

        %;

!
!       Subexpression for Object ID
!

MACRO
        SEM_OBJECT_ID =

$STATE  (SE_OBJECT_ID,
        ((SE_OBJECT_NAM), TPA$_EXIT),
        ((SE_OBJECT_NUM), TPA$_EXIT)
        );

$STATE  (SE_OBJECT_NUM,
        ((SE_OBJ_NUM), TPA$_EXIT, , TRUE, ACT$GL_ADR_Q)
        );

$STATE  (SE_OBJ_NUM,
        (TPA$_LAMBDA, , ACT$CLRLONG, , , ACT$GL_ADR_Q)
        );
$STATE  (,
        (TPA$_DECIMAL, TPA$_EXIT, ACT$NUM_RNG, , ,
                NUM_RANGE (LOW_OBJ_NUM, HIGH_OBJ_NUM))
        );

$STATE  (SE_OBJECT_NAM,
        (TPA$_LAMBDA, , ACT$CLRLONG, , , ACT$GL_ADR_Q)
        );

$STATE  (,
        (TPA$_SYMBOL, TPA$_EXIT, ACT$STR_LEN, , , LEN_OBJ_NAM)
        );

        %;
```

```
!
!       Subexpressions for a query state
!

MACRO
        SEM_QUERY =

$STATE  (SE_QRY_YES,
        ('YES'),
        );

$STATE  (,
        (TPA$_EOS, TPA$_EXIT)
        );

$STATE  (SE_QRY_NO,
        ('NO'),
        );

$STATE  (,
        (TPA$_EOS, TPA$_EXIT)
        );

        %;
```

```
!
!       Subexpressions for load parameters
!
MACRO
        SEM_LOAD (CLS) =

!
!       Subexpression for service device
!
$STATE  (%NAME ('ST_',CLS,'_SDV'),

        KEYWORD_STATE
        (CLS,

        SDVA,  'DA',
        SDVL,  'DL',
        SDVMC, 'DMC',
        SDVP,  'DP',
        SDVQ,  'DQ',
        SDVTE, 'DTE',
        SDVU,  'DU',
        SDVUP, 'DUP',
        SDVKL, 'KL8',
        SDVMP, 'DMP',
        SDVMV, 'DMV',
        SDVPV, 'DPV',
        SDVMF, 'DMF',
        SDVUN, 'UNA',

        )
        );

!
!       Software identification
!
$STATE  (SE_SOFT_ID,
        ( (SE_QUOT_STR), TPA$_EXIT, AC $STR_LEN, , , LEN_SOFT_ID)
        );

!
!       Software type
!
$STATE  (%NAME ('ST_',CLS,'_STY'),

        DISPATCH_STATES
        (CLS,

        STSL, 'SECONDARY',
        STTL, 'TERTIARY',
        STOS, 'SYSTEM',
```

```
        )
        );

$STATE  (%NAME ('ST_', CLS, '_PRC_STSL'),               ! Secondary loader
        ( 'LOADER' ),
        (TPA$_LAMBDA)
        );

$STATE  (,
        ( (%NAME ('ST_',CLS,'_STSL') ), TPA$_EXIT)
        );

$STATE  (%NAME ('ST_', CLS, '_PRC_STTL'),               ! Tertiary loader
        ( 'LOADER' ),
        (TPA$_LAMBDA)
        );

$STATE  (,
        ( (%NAME ('ST_',CLS,'_STTL') ), TPA$_EXIT)
        );

$STATE  (%NAME ('ST_', CLS, '_PRC_STOS'),               ! System
        ( (%NAME ('ST_',CLS,'_STOS') ), TPA$_EXIT)
        );


        SUB_EXPRESSIONS
        (CLS,

        STSL, TPA$_LAMBDA,
        STTL, TPA$_LAMBDA,
        STOS, TPA$_LAMBDA

        )


!
!       Cpu type
!

$STATE  (%NAME ('ST_',CLS,'_CPU'),

        KEYWORD_STATE
        (CLS,

        CPU10, 'DECSYSTEM1020',
        CPU11, 'PDP11',
        CPU8,  'PDP8',
        VAX,   'VAX',

        )
        );

        %;
```

```
!END
!ELUDOM
```

NCPCONCAR
LIS

NCPERRMSG
LIS

NCPCONMAN
LIS

NCPLIBRY
B32

NCPMAIN
LIS

NCPNETIO
LIS

NMAHEAD
B32

NCPLIBRY
LIS

NMATAIL
B32