C 10

```
EEEEEEEEEEEEEEE  DDDDDDDDDDDD     TTTTTTTTTTTTTTT
EEEEEEEEEEEEEEE  DDDDDDDDDDDD     TTTTTTTTTTTTTTT
EEEEEEEEEEEEEEE  DDDDDDDDDDDD     TTTTTTTTTTTTTTT
EEE              DDD         DDD        TTT
EEE              DDD         DDD        TTT
EEE              DDD         DDD        TTT
EEE              DDD         DDD        TTT
EEE              DDD         DDD        TTT
EEE              DDD         DDD        TTT
EEEEEEEEEEEE     DDD         DDD        TTT
EEEEEEEEEEEE     DDD         DDD        TTT
EEEEEEEEEEEE     DDD         DDD        TTT
EEE              DDD         DDD        TTT
EEE              DDD         DDD        TTT
EEE              DDD         DDD        TTT
EEE              DDD         DDD        TTT
EEE              DDD         DDD        TTT
EEE              DDD         DDD        TTT
EEEEEEEEEEEEEEE  DDDDDDDDDDDD           TTT
EEEEEEEEEEEEEEE  DDDDDDDDDDDD           TTT
EEEEEEEEEEEEEEE  DDDDDDDDDDDD           TTT
```

```
EX

Mo
--
ED
ED
ED
ED
ED
ED
ED
ED
ED
ED
ED
ED
ED
ED
ED
ED
ED
ED
ED
ED
SY
LB
LI
```

```
**FILE**ID**EDT                                   L 14



EEEEEEEEEE  DDDDDDDD    TTTTTTTTTT
EEEEEEEEEE  DDDDDDDD    TTTTTTTTTT
EE          DD      DD      TT
EE          DD      DD      TT
EE          DD      DD      TT
EE          DD      DD      TT
EEEEEEEE    DD      DD      TT
EEEEEEEE    DD      DD      TT
EE          DD      DD      TT
EE          DD      DD      TT
EE          DD      DD      TT          ....
EE          DD      DD      TT          ....
EEEEEEEEEE  DDDDDDDD        TT          ....
EEEEEEEEEE  DDDDDDDD        TT          ....


RRRRRRRR    EEEEEEEEEE    QQQQQQ
RRRRRRRR    EEEEEEEEEE    QQQQQQ
RR      RR  EE          QQ      QQ
RR      RR  EE          QQ      QQ
RR      RR  EE          QQ      QQ
RR      RR  EE          QQ      QQ
RRRRRRRR    EEEEEEEE    QQ      QQ
RRRRRRRR    EEEEEEEE    QQ      QQ
RR  RR      EE          QQ  QQ  QQ
RR  RR      EE          QQ  QQ  QQ
RR    RR    EE          QQ    QQ
RR    RR    EE          QQ    QQ
RR      RR  EEEEEEEEEE    QQQQ  QQ
RR      RR  EEEEEEEEEE    QQQQ  QQ
```

```
!+
! This file, EDT.REQ, contains definitions for EDT.
!
! Edit history:
!
! 1-001 - Beginning of edit history.
! 1-002 - Add ASSERT macro, remove bugcheck codes.  JBS 01-Jun-1981
! 1-003 - Offset the PDP-11 error codes. so they can be distinguished
!          from system-specific error codes.  JBS 16-Jul-1981
! 1-004 - Remove the error messages, putting them in
!          ERRMSG.REQ.  JBS 20-Jul-1981
! 1-005 - Add two fields to TBCB ; one  points to the previous buffer,
!          the other marks the buffer as a macro. Delete
!          the creation of the MAC_BLOCK structure TMV 6-Aug-81
! 1-006 - Add the verb number for the new bell verb. STS 10-Aug-1981
! 1-007 - Add INP_JOURNAL and INP_COMMAND to replace INP_FILE.  This
!          lets us journal the responses to SUBSTITUTE/QUERY in
!          the journal file.  JBS 16-Aug-1981
! 1-008 - Add the verb number for the new day/time verb. STS 31-Aug-1981
! 1-009 - Update the routine and variable names.  JBS & TMV 16-Aep-1981
! 1-010 - Add new verbs to set up default verb. STS 21-Sep-1981
! 1-011 - Add new verbs for delete select and toggle select. STS 23-Sep-1981
! 1-012 - Add new search and select verb. STS 24-Sep-1981
! 1-013 - Add literals for word and para types. STS 23-Oct-1981
! 1-014 - Add PREV_RANGE.  JBS 02-Nov-1981
! 1-015 - Add definitions for file i/o codes and streams.  STS 08-Dec-1981
! 1-016 - Change edt$$k to edt$k for file i/o definitions. STS 09-Dec-1981
! 1-017 - Add macro to set up address and length in string desc. STS 11-Jan-1982
! 1-018 - Fix above macro to work with 11's.  STS 13-Jan-1982
! 1-019 - Add literals for open output seq and open output noseq. STS 13-Jan-1982
! 1-020 - Chang string_desc macro for bliss16.  STS 15-Jan-1982
! 1-021 - Change 32-bit arithmetic to 48-bit arithmetic.  SMB 15-Jan-1982
! 1-022 - Modify block allocation so that odd address traps don't occur on 11's.  SMB 25-Jan-1982
! 1-023 - Remove original line numbers.  SMB 29-Jan-1982
! 1-024 - Make callable literals global. STS 08-Mar-1982
! 1-025 - Remove callable literals. STS 08-Mar-1982
! 1-026 - Add symbols for control C handling.  JBS 24-May-1982
! 1-027 - Change VMS multiply.  SMB 25-May-1982
! 1-028 - Add EDT$$K_FMT_BUFLEN.  JBS 05-Jul-1982
! 1-029 - Add verb for xlate. STS 13-Aug-1982
! 1-030 - Remove the keypad definitions to KEYPADDEF.REQ.  JBS 13-Aug-1982
! 1-031 - Add ASC_K_CSI, for 8-bit keyboards.  JBS 17-Aug-1982
! 1-032 - Add ASC_K_SS3, for 8-bit keyboards.  JBS 20-Aug-1982
! 1-033 - Add verb_K_clss. STS 26-Aug-1982
! 1-034 - Add K_RDAHED_LEN.  JBS 31-Aug-1982
! 1-035 - Add new screen data structures.  SMB 11-Sep-1982
! 1-036 - Put back a line that was deleted by mistake.  SMB 15-Sep-1982
! 1-037 - Revise the EDIT section of the new screen data structures.  JBS 17-Sep-1982
! 1-038 - Add CC_RDCNT.  JBS 17-Sep-1982
! 1-039 - Remove CC_RDCNT. STS 20-Sep-1982
! 1-040 - Work on conditionalizing addline macro for speed. STS 30-Sep-1982
! 1-041 - Add memory allocation maximum.  SMB 18-Oct-1982
! 1-042 - Add macros for comparing line numbers. STS 20-Oct-1982
! 1-043 - Work on 11-version of compare macro. STS 21-Oct-1982
! 1-044 - Bind high word of linenumbers in compare macro. STS 21-Oct-1982
! 1-045 - Fix bug in compare. STS 22-Oct-1982
! 1-046 - Work on 11 version of compare macro. STS 26-Oct-1982
! 1-047 - Change 11 compare to call EDT$$CMP_LNO. STS 27-Oct-1982
! 1-048 - Add SCR_EDIT_MINPOS, remove a bunch of unused and obsolete definitions.  JBS 27-Oct-1982
! 1-049 - Reduce the size of the screen edit area on the PDP-11.  This saves
!           space at the expense of time.  JBS 15-Nov-1982
! 1-050 - Remove the edit buffer entirely.  JBS 27-Dec-1982
! 1-051 - Reduce the amount of code generated by the ASSERT macro, to try
!           to save space on the PDP-11.  JBS 16-Jan-1983
! 1-052 - Correct the definition of SS3.  JBS 19-Jan-1983
! 1-053 - Change the format buffer size for VMS.  SMB 24-Feb-1983
! 1-054 - Remove WC_K_NUM_BUKT.  JBS 29-Mar-1983
!-
```

```
!+
! DEFINITION_DEFINITIONS
! ----------------------
!
! The following definitions are used to facilitate further definitions.
!-

!+
! Field definition macros.  This set of macros allows for definitions
! of the fields of data structures, letting the compiler compute the
! the offsets.
!-

COMPILETIME FIELD_OFFSET = 0;
COMPILETIME NUMBER_ONE = 1;

MACRO START_FIELDS(FIELD_NAME) =
  FIELD FIELD_NAME =
  SET
  %ASSIGN(FIELD_OFFSET,0) %;

MACRO A_FIELD(FIELD_NAME1,LENGTH) =
  FIELD_NAME1 = [FIELD_OFFSET/8,FIELD_OFFSET MOD 8,LENGTH,0]
  %ASSIGN(FIELD_OFFSET,FIELD_OFFSET+LENGTH) %;

MACRO INC_FIELD (LENGTH) =
  %ASSIGN(FIELD_OFFSET,FIELD_OFFSET+LENGTH) %;

MACRO END_FIELDS = TES;%;

MACRO STRUC_SIZE(SIZE) = LITERAL SIZE = (FIELD_OFFSET+7)/8; %;
```

```
!+
! IMPLEMENTATION PARAMETERS.
!
! The following definitions are parameters used in the work-file system
! which may require re-definition for different implementations.
!-


LITERAL
  WF_BLN_LEN            = 16,   ! Bit length of a work-file block number.
  LINE_NUM_LEN          = 16;   ! Bit length of a line number. (actually 3*16=48)
```

```
!+
! TBCB_DEFINITION
! ---------------
!
!       The EDT work file can contain multiple, independent data sets
! referred to as Text Buffers.  A text buffer corresponds to the construct
! of the same name found in EDT user documentation, it is a seqential file
! of variable length records.  The records are grouped together into blocks
! of 512 characters.  The records in a block are sequentially ordered,
! though the blocks themselves are not.  Each block contains a two-byte
! link to the previous and following blocks.  In addition to the lines in
! the work-file, an input file may be associated with a text buffer.  In this
! case the input file is logically placed at the end of the text buffer.
! The Text buffer is accessed via a control block called the Text Buffer
! Control Block, or TBCB.
!-

START_FIELDS(TBCB_FIELDS)
  A_FIELD(TBCB_LINE_ADDR,%BPADDR),      ! Pointer to current line.
  A_FIELD(TBCB_CUR_BUKT,WF_BLN_LEN),    ! Current bucket number.
  A_FIELD(TBCB_CUR_LIN,LINE_NUM_LEN),   ! Current line number.
  A_FIELD(TBCB_CUR_LINM,LINE_NUM_LEN),
  A_FIELD(TBCB_CUR_LINH,LINE_NUM_LEN),
  A_FIELD(TBCB_CHAR_POS,WF_BLN_LEN),     ! The character position within the line
  A_FIELD(TBCB_FIRST_BUKT,WF_BLN_LEN),  ! First bucket number.
  A_FIELD(TBCB_LAST_BUKT,WF_BLN_LEN),   ! Last bucket number.
  A_FIELD(TBCB_INPUT_LINE,LINE_NUM_LEN),! Number of last input line.
  A_FIELD(TBCB_INPUT_LINM,LINE_NUM_LEN),
  A_FIELD(TBCB_INPUT_LINH,LINE_NUM_LEN),
  A_FIELD(TBCB_LINE_COUNT,LINE_NUM_LEN),! Count of lines in buffer.
  A_FIELD(TBCB_LC_M,LINE_NUM_LEN),
  A_FIELD(TBCB_LC_H,LINE_NUM_LEN),
  A_FIELD(TBCB_CHAR_COUNT,%BPVAL),      ! Count of chars in buffer.
  A_FIELD(TBCB_PREV_BUF,%BPADDR),       ! Pointer to previous text buffer.
  A_FIELD(TBCB_NEXT_BUF,%BPADDR),       ! Pointer to next text buffer.
  A_FIELD(TBCB_INPUT_RAB,8),            ! Pointer to input RAB.
  A_FIELD(TBCB_IS_MAC,8),               ! This buffer is a macro
  A_FIELD(TBCB_NAME_LEN,8),             ! Length of buffer name.
  A_FIELD(TBCB_NAME,0)                  ! Name of buffer

END_FIELDS

STRUC_SIZE(TBCB_SIZE)                   ! Define size of TBCB.

MACRO TBCB_BLOCK = BLOCK[TBCB_SIZE,BYTE] FIELD(TBCB_FIELDS)% ;

!+
! The pos block is the portion of the TBCB which contains information needed
! to locate the current line.  This block must be identical to the first
! part of the TBCB or everything will fail.
!-

START_FIELDS(POS_FIELDS)
  A_FIELD(POS_LINE_ADDR,%BPADDR),       ! Pointer to current line.
  A_FIELD(POS_CUR_BUKT,WF_BLN_LEN),     ! Current bucket number.
  A_FIELD(POS_CUR_LIN,LINE_NUM_LEN),      ! Current line number.
  A_FIELD(POS_CUR_LINM,LINE_NUM_LEN),
  A_FIELD(POS_CUR_LINH,LINE_NUM_LEN),
  A_FIELD(POS_CHAR_POS,WF_BLN_LEN)
END_FIELDS

STRUC_SIZE(POS_SIZE)                      ! Define size of position information

MACRO POS_BLOCK = BLOCK[POS_SIZE,BYTE] FIELD(POS_FIELDS)%;
```

```
!+
! TEXT LINE DEFINITIONS
!
! A line number contains an integer part and a fractional part.
!-

START_FIELDS(LIN_FIELDS)
  A_FIELD(LIN_LENGTH,8),                 ! Length of line
  A_FIELD(LIN_NUM,LINE_NUM_LEN),         ! The line number
  A_FIELD(LIN_NUMM,LINE_NUM_LEN),
  A_FIELD(LIN_NUMH,LINE_NUM_LEN),
  A_FIELD(LIN_TEXT,0)                    ! The actual text
END_FIELDS

STRUC_SIZE(LIN_FIXED_SIZE)

MACRO LIN_BLOCK = BLOCK[LIN_FIXED_SIZE,BYTE] FIELD(LIN_FIELDS)%;
```

```
!+
! WORK-FILE BUCKET DEFINITIONS
! ----------------------------
!
! The work file is orgainized into blocks of WF_BLOCK_SIZE characters.
! Each Text Buffer in the work file consists of a linked list of blocks.
!-

LITERAL WF_BUKT_SIZE = 512;             ! Size of a work-file block

START_FIELDS(WFB_FIELDS)
  A_FIELD(WFB_PREV_BUKT,WF_BLN_LEN),    ! Number of previous bucket
  A_FIELD(WFB_NEXT_BUKT,WF_BLN_LEN),    ! Number of next bucket
  A_FIELD(WFB_END,%BPVAL),              ! Offset to last record in block
  A_FIELD(WFB_RECORDS,0)                ! Address of first record in block
END_FIELDS

STRUC_SIZE(WFB_FIXED_SIZE)
```

```
!+
! LINE NUMBER BLOCK DEFINITIONS
!
! The line number is defined as a block, so it can be handled as
! three 16-bit words.
!-

FIELD LN_FIELDS =
    SET
    LN_LO = [0,0,16,0],
    LN_MD = [2,0,16,0],
    LN_HI = [4,0,16,0]
    TES;

MACRO LN_BLOCK = BLOCK[6,BYTE] FIELD(LN_FIELDS) %;

LITERAL LN_SIZE = 6;

STRUCTURE
    LNOVECTOR[I;N] = [N*LN_SIZE] (LNOVECTOR+I*LN_SIZE);
```

```
!+
! Semantic node definitions.
!
! The following defines the structures created by the EDT
! command parser semantic routines.  These structures form
! a tree-like representation of the command.
!
! The fields which are grouped together are re-definitions of the
! same slot for use in different types of nodes.
!-

FIELD NODE_FIELDS =
    SET
    NODE_TYPE    = [0,0,8,0],                  ! Identifies the type of node

    COM_NUM      = [1,0,8,0],                  ! Identifies the command
    RAN_TYPE     = [1,0,8,0],                  ! Identifier type of range
    OP_TYPE      = [1,0,8,0],                  ! Identifies type of operand
    SEQ_VAL      = [1,0,8,0],                  ! Did the seq switch have value.

    RANGE1       = [%UPVAL,0,%BPVAL,0],        ! First range specifier
    RAN_VAL      = [%UPVAL,0,%BPVAL,0],        ! Value for range specifier
    SW_BITS      = [%UPVAL,0,%BPVAL,0],        ! Bits for each possible switch
    SRCHADDR     = [%UPVAL,0,%BPVAL,0],        ! Address of search string
    SET_TYPE     = [%UPVAL,0,%BPVAL,0],        ! Which type of set command
    LEFT_OP      = [%UPVAL,0,%BPVAL,0],        ! Left operand for binary ops
    OP_LEN       = [%UPVAL,0,%BPVAL,0],        ! operand length for op nodes.
    OP_VAL       = [%UPVAL,0,%BPVAL,0],        ! Operand value for numerics.
    COM_EXPR     = [%UPVAL,0,%BPVAL,0],        ! Expression pointer for LET
    OP_LEFTOP    = [%UPVAL,0,%BPVAL,0],        ! Left operand for operators.
    SUB_BASE     = [%UPVAL,0,%BPVAL,0],        ! Substring base string.

    RANGE2       = [%UPVAL*2,0,%BPVAL,0],      ! Second range specifier
    SUB_RANGE    = [%UPVAL*2,0,%BPVAL,0],      ! Pointer to range for ranges
    STR_PNT      = [%UPVAL*2,0,%BPVAL,0],      ! Pointer to a search string
    SRCHLEN      = [%UPVAL*2,0,%BPVAL,0],      ! Search string length
    FILSPEC      = [%UPVAL*2,0,%BPVAL,0],      ! File specification address
    SW_VAL1      = [%UPVAL*2,0,%BPVAL,0],      ! First value for switches
    AS_STR       = [%UPVAL*2,0,%BPVAL,0],      ! Addr of string for AS
    RIGHT_OP     = [%UPVAL*2,0,%BPVAL,0],      ! Right operand for binary ops.
    BUF_NAME     = [%UPVAL*2,0,%BPVAL,0],      ! Address of buffer name
    OP_ADDR      = [%UPVAL*2,0,%BPVAL,0],      ! Operand address for op nodes.
    COM_VARBL    = [%UPVAL*2,0,%BPVAL,0],      ! Variable pointer for LET
    OP_RIGHTOP   = [%UPVAL*2,0,%BPVAL,0],      ! Right operand for operators.
    SUB_START    = [%UPVAL*2,0,%BPVAL,0],      ! Substring start pos.
    TAB_COUNT    = [%UPVAL*2,0,%BPVAL,0],      ! Count for tabs adjust.

    SET_VAL1     = [%UPVAL*3,0,%BPVAL,0],      ! Value for set command
    REPADDR      = [%UPVAL*3,0,%BPVAL,0],      ! Replace string address
    FSPCLEN      = [%UPVAL*3,0,%BPVAL,0],      ! File spec length
    AS_LEN       = [%UPVAL*3,0,%BPVAL,0],      ! Length of string for AS
    BUF_LEN      = [%UPVAL*3,0,%BPVAL,0],      ! length of buffer name
    SUB_LENGTH   = [%UPVAL*3,0,%BPVAL,0],      ! Substring length.

    NEXT_COM     = [%UPVAL*4,0,%BPVAL,0],      ! Pointer to next command
    NEXT_RANGE  = [%UPVAL*4,0,%BPVAL,0],        ! Pointer to next range
    REPLEN      = [%UPVAL*4,0,%BPVAL,0],        ! Replace string length
    SET_VAL     = [%UPVAL*4,0,%BPVAL,0],
    KEY_VAL     = [%UPVAL*4,0,%BPVAL,0],        ! Number of key for def key

    PREV_RANGE  = [%UPVAL*5,0,%BPVAL,0],        ! Reverse of NEXT_RANGE
    SWITS       = [%UPVAL*5,0,%BPVAL,0],        ! Switch block pointer
    SW_VAL2     = [%UPVAL*5,0,%BPVAL,0],        ! Second option switch value

    SW_OVR1     = [%UPVAL*6,0,%BPVAL,0],        ! Part of second option switch

    SW_OVR2     = [%UPVAL*7,0,%BPVAL,0]         ! Part of second option switch
    TES;

LITERAL
    NUM_NODES   = 20,                           ! Number of semantic nodes
    NODE_SIZE   = 8*%UPVAL;                     ! Size of semantic node

LITERAL ! Node type equates

    COM_NODE    = 1,                            ! Command node
    RANGE_NODE  = 2,                            ! Range node
    STR_NODE    = 3,                            ! SUBSTITUTE strings
    SW_NODE     = 4,                            ! Option switch value
    OP_NODE     = 5;                            ! Expression operand

MACRO NODE_BLOCK = BLOCK[NODE_SIZE,BYTE] FIELD(NODE_FIELDS) %;
```

```
!+
! ASCII CHARACTER DEFINITIONS
!
! Commonly used non-printing ASCII characters.
!-

LITERAL
    ASC_K_BS     = %O'10',
    ASC_K_TAB    = %O'11',
    ASC_K_LF     = %O'12',
    ASC_K_CTRL_K= %O'13',
    ASC_K_FF     = %O'14',
    ASC_K_CR     = %O'15',
    ASC_K_SO     = %O'16',
    ASC_K_SI     = %O'17',
    ASC_K_CTRL_U= %O'25',
    ASC_K_CTRL_Z= %O'32',
    ASC_K_ESC    = %O'33',
    ASC_K_SP     = %O'40',
    ASC_K_DEL    = %O'177',
    ASC_K_CSI    = ASC_K_ESC + %X'80',
    ASC_K_SS3    = ASC_K_SI + %X'80';
```

```
!+
! COMMAND NUMBER DEFINITIONS
!
! The following values are used in a command type node to specify which
! command it is.
!-

LITERAL
    COM_NULL    = 0,
    COM_CHANGE  = 1,
    COM_COPY    = 2,
    COM_DEFINE  = 3,
    COM_DELETE  = 4,
    COM_EXIT    = 5,
    COM_FIND    = 6,
    COM_INCLUDE = 7,
    COM_INSERT  = 8,
    COM_MOVE    = 9,
    COM_PRINT   = 10,
    COM_QUIT    = 11,
    COM_REPLACE = 12,
    COM_RESEQ   = 13,
    COM_SET     = 14,
    COM_SHOW    = 15,
    COM_SUBS    = 16,
    COM_TYPE    = 17,
    COM_WRITE   = 18,
    COM_SUBS_NEXT=19,
    COM_HELP    = 20,
    COM_CLEAR   = 21,
    COM_TADJ    = 22,
    COM_FILL    = 23,
    COM_DEF_MAC = 24,
    COM_MAC_CALL= 25,
!   COM_VERIFY  = ?,
    LAST_COM    = 25;
```

```
!+
! RANGE TYPE DEFINITIONS
!
! The following constants are used in range nodes to specify the type of
! range.
!-

LITERAL
    RAN_NULL    = 0,
    RAN_NUMBER  = 1,
    RAN_DOT     = 2,
    RAN_STR     = 3,
    RAN_BEGIN   = 4,
    RAN_END     = 5,
    RAN_ORIG    = 6,
    RAN_PATTERN = 7,
    RAN_LAST    = 8,
    RAN_BEFORE  = 9,
    RAN_REST    = 10,
    RAN_WHOLE   = 11,
    RAN_SELECT  = 12,
    RAN_BUFFER  = 13,
    RAN_PLUS    = 14,
    RAN_MINUS   = 15,
    RAN_FOR     = 16,
    RAN_THRU    = 17,
    RAN_MINSTR  = 18,
    RAN_ALL     = 19,
    RAN_AND     = 20,
    NUM_RAN     = 20,    ! Total number of ranges
    NUM_SLR     = 7;     ! number of single line ranges

!
! Operand types for operand nodes.
!
LITERAL
    OP_STRING   = 0,     ! Operand is a quoted string
    OP_NUM      = 1,     ! Operand is a number
    OP_VAR      = 2,     ! Operand is a variable
    OP_DOT      = 3,     ! Operand is the dot pseudo variable
    OP_ADD      = 4,     ! Operand is an addition operator
    OP_SUB      = 5,     ! Operand is a subtractions operator
    OP_MULT     = 6,     ! Operand is a multiplication operator
    OP_DIV      = 7,     ! Operand is a division operator
    OP_AND      = 8,     ! logical and
    OP_OR       = 9,     ! logical or
    OP_LSS      = 10,    ! compare for less
    OP_LEQ      = 11,    ! compare for less or equal
    OP_EQL      = 12,    ! Compare for equality
    OP_GEQ      = 13,    ! compare for greater or equal
    OP_GTR      = 14,    ! compare for greater
    OP_NEQ      = 15,    ! compare for not equal
    OP_AMP      = 16,    ! concatenation
    OP_SUBSTR   = 17,    ! substringing
    OP_NEG      = 18,    ! negation
    OP_NOT        = 19,    ! logical not
    OP_LENGTH     = 20,    ! length of
    OP_COL        = 21,    ! current column
    OP_FIND       = 22,
    OP_POS        = 23,    ! current position
    OP_LAST_OP    = 23;    ! last operand type
```

```
 !+
 ! LINE NUMBER HANDLING MACROS
 !
 ! These macros are used for arithmetic involving line numbers, so it can
 ! be transportable across systems with various word lengths.  At least 48
 ! bits of precision are required for line numbers.  Line numbers are stored
 ! as an integer with a scale of -5, i.e. the true value * 10**5, so we can
 ! have 5 decimal positions and 10 integer positions in the line number.
 !-

%IF %BLISS(BLISS32) %THEN
MACRO
    ADDLINE(S1,S2,DEST,MAX) =
 !+
 ! Add 2 48-bit numbers using 2 longwords (so we can
 ! use the BLISS-32 Built-in macros).
 !-
    BEGIN
%IF %CTCE(S1) %THEN
    %IF %LENGTH EQL 2 %THEN
 !+
 ! add a compile time expression to s2 and store it in s2
 !-
        BEGIN
        BIND
            FIRST_LWORD = S2 :LONG,
            NEXT_WORD = (S2+4) : WORD;
        FIRST_LWORD = .FIRST_LWORD +S1;
        IF .FIRST_LWORD LSSU S1
        THEN
            NEXT_WORD = .NEXT_WORD + 1;
        END
    %ELSE
 !+
 ! add a compile time expression to s2 and store it in dest
 !-
        BEGIN
        BIND FIRST_WORD = (DEST) : LONG,
             NEXT_WORD = (DEST+4) : WORD,
             SOURCE_2LO = (S2) : LONG,
             SOURCE_2HI = (S2+4) : WORD;

        FIRST_WORD = .SOURCE_2LO + S1;
        IF (.FIRST_WORD LSSU S1)
        THEN
            NEXT_WORD = .SOURCE_2HI + 1
        ELSE
            NEXT_WORD = .SOURCE_2HI;
        END
    %FI
%ELSE
 !+
 ! we don't have a compile time expression, but we are adding two 48-bit numbers
 !-

    %IF %LENGTH EQL 2 %THEN              ! store the result in S2
```

```
        BEGIN
        LOCAL SAVE: WORD;
        BUILTIN ADDM;
        BIND UPPER_WORD = (S2+6) : WORD;
        SAVE = .UPPER_WORD;
        ADDM(2,S1,S2,S2);
        UPPER_WORD = .SAVE;
        END
    %ELSE
        %IF %LENGTH EQL 3 %THEN         ! store the result in DEST
        BEGIN
        LOCAL
            SAVE : WORD;
        BUILTIN ADDM;
        BIND UPPER_WORD = (DEST+6) : WORD;

        SAVE = .UPPER_WORD;
        ADDM(2,S1,S2, DEST);
        UPPER_WORD = .SAVE;
        END
        %ELSE                           ! store the result in DEST and return
            BEGIN                       ! any overflow in MAX
            LOCAL
                SAVES2 : WORD,
                SAVED : WORD;
            BIND
                S1_UP = (S1+6) : WORD,
                S2_UP = (S2+6) : WORD,
                DEST_UP = (DEST+6) : WORD;

            BUILTIN ADDM;
            SAVES2 = .S2_UP + .S1_UP;
            SAVED = .DEST_UP;
            ADDM(2,S1,S2,DEST);

!+
! Get the overflow bit
!-
            IF .DEST_UP EQL .SAVES2
            THEN
                MAX = 0
            ELSE
                MAX = 1;
            DEST_UP = .SAVED;
            END
        %FI
    %FI
%FI
    END%,


    SUBLINE(S1,S2,DEST) =
!+
! Subtract 2 48-bit numbers using 2 longwords
!-
    BEGIN
```

```
%IF %CTCE(S1) %THEN
    %IF %LENGTH EQL 2 %THEN
!+
! we have a compile time expression to add to S2 and store in S2
!-
        BEGIN
        LOCAL SAVE : LONG;
        BIND
            FIRST_WORD = S2 :LONG,
            NEXT_WORD = (S2+4) : WORD;
        SAVE = .FIRST_WORD;
        FIRST_WORD = .FIRST_WORD - S1;
        IF .FIRST_WORD GTRU .SAVE
        THEN
            NEXT_WORD = .NEXT_WORD - 1;
        END
    %ELSE
!+
! add the compile time expression to S2 and store it in DEST
!-
        BEGIN
        BIND FIRST_WORD = (DEST) : LONG,
             NEXT_WORD = (DEST+4) : WORD,
             SOURCE_2LO = (S2) : LONG,
             SOURCE_2HI = (S2+4) : WORD;

        FIRST_WORD = .SOURCE_2LO - S1;
        IF .FIRST_WORD GTRU .SOURCE_2LO
        THEN
            NEXT_WORD = .SOURCE_2HI - 1
        ELSE
            NEXT_WORD = .SOURCE_2HI;
        END
    %FI
%ELSE
    %IF %LENGTH EQL 2 %THEN
!+
! add two 48 bit numbers and store result in S2
!-
        BEGIN
        LOCAL SAVE: WORD;
        BUILTIN SUBM;
        BIND UPPER_WORD = (S2+6) : WORD;
        SAVE = .UPPER_WORD;
        SUBM(2,S1,S2,S2);
        UPPER_WORD = .SAVE;
        END
    %ELSE
!+
! add two 48 bit numbers and store result in DEST
!-
        BEGIN
        LOCAL
            SAVE : WORD;
        BUILTIN SUBM;
        BIND UPPER_WORD = (DEST+6) : WORD;
        SAVE = .UPPER_WORD;
        SUBM(2,S1,S2, DEST);
        UPPER_WORD = .SAVE;
        END
    %FI
%FI
    END%,


    MULTLINE(S1,S2,DEST) =
!+
! Multiply 2 48-bit numbers, but S1 MUST be <= 100,000
!-
        BEGIN
        BIND
            M1 = S1 : BITVECTOR [32];

        LOCAL M2 : VECTOR[2],
              P  : VECTOR[2];
        BUILTIN ADDM, ASHQ;
!+
! Set up the multiplicand and result in 64 bits, zeroeing
! out the upper 16-bits.
!-
        M2[0] = .(S2)<0,32>; M2[1] = .(S2+4)<0,16>;
        P[0] = 0; P[1] = 0;
!+
! Since 65535 < multiplier <+ 100,000... we only need to
! examine the low order 17-bits.
!-
        DECR I FROM 16 TO 0
        DO
            BEGIN
            ASHQ(%REF(1), P, P);                    ! Shift result left by 1 (multiply by 2)
            IF (.M1[.I]) THEN  ADDM(2, P, M2, P);   ! Add multiplicand to result
            END;                                    ! if multiplier bit set
        (DEST)<0,32> = .P[0];  (DEST+4)<0,16> = .P[1];
        END%,

!+
! compare two 48 bit line numbers to see if they are equal
!-
    LINNOEQL(LIN1,LIN2) =
    BEGIN
    BIND
        NO_1 = LIN1 : VECTOR[3,WORD],
        NO_2 = LIN2 : VECTOR[3,WORD],
        LOW_1 = NO_1[0] : LONG,
        LOW_2 = NO_2[0] : LONG,
        HIGH_1 = NO_1[2] : WORD,
        HIGH_2 = NO_2[2] : WORD;

    IF ((.LOW_1 EQL .LOW_2) AND (.HIGH_1 EQL .HIGH_2))
    THEN
        (1)
    ELSE
        (0)
    END%,

    CMPLNO(LIN1,LIN2) =
    BEGIN
    BIND
        NO_1 = LIN1 : VECTOR[3,WORD],
        NO_2 = LIN2 : VECTOR[3,WORD],
        LOW_1 = NO_1[0] : LONG,
        LOW_2 = NO_2[0] : LONG,
        HIGH_1 = NO_1[2] : WORD,
        HIGH_2 = NO_2[2] : WORD;

        IF (.HIGH_1 LSSU .HIGH_2)
        THEN
            (-1)
        ELSE
            BEGIN
            IF (.HIGH_1 EQL .HIGH_2)
            THEN
                IF (.LOW_1 LSSU .LOW_2)
                THEN
                    (-1)
                ELSE
                    IF (.LOW_1 EQL .LOW_2) THEN (0) ELSE (1)
            ELSE
                (1)
            END
    END%,

    MOVELINE(S,D) = (CH$MOVE(6,S,D))%,                  ! Move 6 bytes of storage

    BUILDLINE(S,D) = (D = S; (D+4) = 0)%;               ! Build a number

!
%ELSE %IF %BLISS(BLISS16) %THEN

MACRO
    ADDLINE(S1,S2,DEST,MAX) =
    BEGIN
    %IF %CTCE(S1) %THEN
        %IF %LENGTH EQL 2 %THEN
!+
! we are addng a constant to source_2 and storing in source_2
!-
                   BEGIN
                   BIND
                        FIRST_WORD = S2:WORD,
                        NEXT_WORD = (S2+2) : WORD,
                        HIGH_WORD = (S2+4) : WORD;
                    FIRST_WORD = .FIRST_WORD + S1;
                    IF .FIRST_WORD EQL 0
                    THEN
                        BEGIN
                        NEXT_WORD = .NEXT_WORD + 1;
                        IF .NEXT_WORD EQL 0 THEN HIGH_WORD = .HIGH_WORD + 1;
                        END;
                   END
        %ELSE
!+
! destination is DEST and we have a compile time constant
!-
        BEGIN
        BIND
            SOURCE_1 = S2 : WORD,
            SOURCE_2 = (S2+2) : WORD,
            SOURCE_3 = (S2+4) : WORD,
            FIRST_WORD = DEST : WORD,
            NEXT_WORD = (DEST+2) : WORD,
            HIGH_WORD = (DEST+4) : WORD;
        FIRST_WORD = .SOURCE_1 + S1;
        NEXT_WORD = .SOURCE_2;
        HIGH_WORD = .SOURCE_3;
        IF .FIRST_WORD EQL 0
        THEN
            BEGIN
            NEXT_WORD = .NEXT_WORD + 1;
            IF .NEXT_WORD EQL 0
            THEN
                HIGH_WORD = .HIGH_WORD + 1 ;
            END;
        END
    %FI
!+
! we don't have a constant
!-

    %ELSE
        %IF %LENGTH EQL 2 %THEN
        BEGIN EXTERNAL ROUTINE A48_ADD; A48_ADD(S1,S2,S2) END
        %ELSE
            %IF %LENGTH EQL 3 %THEN
            BEGIN EXTERNAL ROUTINE A48_ADD; A48_ADD(S1,S2,DEST) END
            %ELSE
                BEGIN EXTERNAL ROUTINE A48_ADD; MAX = A48_ADD(S1,S2,DEST) END
            %FI
        %FI
    %FI
    END%,
    SUBLINE(S1,S2,DEST) =
    BEGIN
    %IF %CTCE(S1) %THEN
        BEGIN
        %IF %LENGTH EQL 2 %THEN
            BEGIN
            LOCAL SAVE : WORD;
            BIND
                FIRST_WORD = S2 : WORD,
                NEXT_WORD = (S2+2) : WORD,
                HIGH_WORD = (S2+4) : WORD;
            SAVE = .FIRST_WORD;
            FIRST_WORD = .FIRST_WORD - S1;
            IF .FIRST_WORD GTRU .SAVE
            THEN
                BEGIN
                NEXT_WORD = .NEXT_WORD - 1;
                IF .NEXT_WORD EQL -1 THEN HIGH_WORD = .HIGH_WORD - 1;
                END;
            END
        %ELSE
!+
! subtract a compile time constant to S2 and put result in DEST
!-
            BEGIN
            BIND
                FIRST_WORD = DEST : WORD,
                NEXT_QORD = (DEST+2) : WORD,
                HIGH_WORD = (DEST+4) : WORD,
                S2_LO = S2 : WORD,
                S2_M = (S2+2) : WORD,
                S2_HI = (S2+4) : WORD;

            FIRST_WORD = .S2_LO - S1;
            NEXT_QORD = .S2_M;
            HIGH_WORD = .S2_HI;
            IF .FIRST_WORD GTRU .S2_LO
            THEN
                BEGIN
                NEXT_WORD = .NEXT_WORD - 1;
                IF .NEXT_WORD EQL -1
                THEN
                    HIGH_WORD = .HIGH_WORD - 1;
                END;
            END
        %FI
        END
    %ELSE
!+
! We don't have a compile time expression
!-
        %IF %LENGTH EQL 2 %THEN
            BEGIN EXTERNAL ROUTINE A48_SUB; A48_SUB(S1,S2,S2) END
        %ELSE
            BEGIN EXTERNAL ROUTINE A48_SUB; A48_SUB(S1,S2,DEST) END
        %FI
    %FI
    END%,
    MULTLINE(S5,S6,D3) =
        BEGIN EXTERNAL ROUTINE A48_MUL; A48_MUL(S5,S6,D3) END %,

    LINNOEQL (LIN1,LIN2) = (CH$EQL(6,LIN1,6,LIN2))%,

    CMPLNO (LIN1,LIN2) =
        BEGIN EXTERNAL ROUTINE EDT$$CMP_LNO; EDT$$CMP_LNO(LIN1,LIN2) END %,

    MOVELINE(S11,D6) = (CH$MOVE(6,S11,D6))%,
    BUILDLINE(S12,D7) = (D7 = S12; (D7+2) = 0; (D7+4) = 0)%;
```

```
%FI %FI
```

```
!+
! OPTION SWITCH BIT DEFINITIONS
!-

LITERAL
    OPT_QUERY   =   2,
    OPT_BRIEF   =   4,
    OPT_NOTYP   =   8,
    OPT_SEQ     =  16,
    OPT_DUPL    =  32,
    OPT_SAVE    =  64,
    OPT_STAY    = 128;

MACRO
    OPB_QUERY   = 1,1 %,
    OPB_BRIEF   = 2,1 %,
    OPB_NOTYP   = 3,1 %,
    OPB_SEQ     = 4,1 %,
    OPB_DUPL    = 5,1 %,
    OPB_SAVE    = 6,1 %,
    OPB_STAY    = 7,1 %;
```

```
! Input source definitions.
!
! These constants define the source command line input.
!
LITERAL
    INP_TERM    = 0,            ! Terminal
    INP_MACRO   = 1,            ! A macro
    INP_COMMAND = 2,            ! The startup file
    INP_JOURNAL = 3;            ! The journal file (only during /RECOVER)

!+
! Terminal type definitions.
!
! These literals define the type of terminal we are running on.
!
LITERAL
    TERM_UNKNOWN= 0,
    TERM_VT52   = 1,
    TERM_VT100  = 2,
    TERM_HCPY   = 3;
!+
! Length of the type-ahead buffer
!-
LITERAL
        K_RDAHED_LEN = 32;

!
! Editor mode definitions.
!
LITERAL
    CHANGE_MODE         = 0,
    LINE_MODE           = 1;

!
! definitions for types of words and paras
!
LITERAL
    DELIMITED           = 0,
    NOT_DELIMITED       = 1,
    WPSPARA             = 0,
    EDTPARA             = 1;
```

```
!+
! Define the error codes.
!-
REQUIRE 'EDTSRC:ERRMSG.REQ';
```

```
!+
! Definition of the screen update data structure.
!
! This structure has an entry for each line which is represented on the screen.
! In NOTRUNCATE mode, each record may occupy one or more screen lines.
!-

START_FIELDS(SCR_FIELDS)
    A_FIELD(SCR_PRV_LINE,%BPADDR),          ! Pointer to the previous line
    A_FIELD(SCR_NXT_LINE,%BPADDR),          ! Pointer to the next line
    A_FIELD(SCR_LINE_IDX,8),                ! The i'th screen line of this record
    A_FIELD(SCR_CHR_FROM,8),                ! Workfile char position from
    A_FIELD(SCR_CHR_TO,8),                  ! Workfile char position to
    A_FIELD(SCR_EDIT_MINPOS,8),             ! Minimum oosition that has had an edit
    A_FIELD(SCR_EDIT_MAXPOS,8),             ! Maximum position that has had an edit
    A_FIELD(SCR_EDIT_FLAGS,8)               ! Modify, delete and insert flags
END_FIELDS

STRUC_SIZE(SCR_SIZE);

MACRO
    SCREEN_LINE = BLOCK[SCR_SIZE,BYTE] FIELD(SCR_FIELDS) %;
!+
! These flags go in SCR_EDIT_FLAGS and are also used when calling EDT$$MRK_LNCHG.
!-
LITERAL
    SCR_EDIT_MODIFY = 1,                    ! This line has been modified
    SCR_EDIT_INSLN = 2,                     ! This line has been inserted
    SCR_EDIT_DELLN = 4;                     ! This line has been deleted
```

```
!+
! This hack added to get around problem in CH$DIFF in BLISS16.
!-

%IF %BLISS(BLISS16) OR %BLISS(BLISS32) %THEN
    MACRO
        CH$PTR_GTR(P1,P2) = (P1) GTRA (P2) %,
        CH$PTR_GEQ(P1,P2) = (P1) GEQA (P2) %,
        CH$PTR_EQL(P1,P2) = (P1) EQLA (P2) %,
        CH$PTR_LEQ(P1,P2) = (P1) LEQA (P2) %,
        CH$PTR_LSS(P1,P2) = (P1) LSSA (P2) %,
        CH$PTR_NEQ(P1,P2) = (P1) NEQA (P2) %;
%ELSE
    MACRO
        CH$PTR_GTR(P1,P2) = CH$DIFF(P1,P2) GTR 0 %,
        CH$PTR_GEQ(P1,P2) = CH$DIFF(P1,P2) GEQ 0 %,
        CH$PTR_EQL(P1,P2) = CH$DIFF(P1,P2) EQL 0 %,
        CH$PTR_LEQ(P1,P2) = CH$DIFF(P1,P2) LEQ 0 %,
        CH$PTR_LSS(P1,P2) = CH$DIFF(P1,P2) LSS 0 %,
        CH$PTR_NEQ(P1,P2) = CH$DIFF(P1,P2) NEQ 0 %;
%FI
```

```
!+
! Define the entity types.
!-

LITERAL

    ENT_K_CHAR  = 1,
    ENT_K_WORD  = 3,
    ENT_K_BW    = 5,
    ENT_K_EW    = 7,
    ENT_K_LINE  = 9,
    ENT_K_BL    = 11,
    ENT_K_NL    = 13,
    ENT_K_VERT  = 15,
    ENT_K_EL    = 17,
    ENT_K_SEN   = 19,
    ENT_K_BSEN  = 21,
    ENT_K_ESEN  = 23,
    ENT_K_PAR   = 25,
    ENT_K_BPAR  = 27,
    ENT_K_EPAR  = 29,
    ENT_K_PAGE  = 31,
    ENT_K_BPAGE = 33,
    ENT_K_EPAGE = 35,
    ENT_K_BR    = 37,
    ENT_K_ER    = 39,
    ENT_K_QUOTE = 41,
    ENT_K_SR    = 43,
    LAST_K_ENT  = 43;
```

```
!+
! Define the verb numbers.
!
! These are the codes used to represent the change mode subcommands.
!
! The verbs from VERB_MOVE through VERB_APPEND require entities and
! their verb numbers must remain contiguous.
!-

LITERAL

    VERB_K_MOVE =       0,
    VERB_K_DELETE=      1,
    VERB_K_REPLACE=     2,
    VERB_K_CHGC =       3,
    VERB_K_CHGU =       4,
    VERB_K_CHGL =       5,
    VERB_K_SSEL =       6,
    VERB_K_FILL =       7,
    VERB_K_TADJ =       8,
    VERB_K_CUT  =       9,
    VERB_K_APPEND=      10,

    VERB_K_SEL  =       11,
!+
! verbs verb_k_subs through verb_k_cc are special since they
! require variable length strings - keep them together with
! subs always first and cc last.
!-
    VERB_K_SUBS =       12,
    VERB_K_PASTE=       13,
    VERB_K_INSERT=      14,
    VERB_K_XLATE =      15,
    VERB_K_CC   =       16,
    VERB_K_EXIT =       17,
    VERB_K_SN   =       18,
    VERB_K_UNDC =       19,
    VERB_K_UNDW =       20,
    VERB_K_UNDL =       21,
    VERB_K_ADV  =       22,
    VERB_K_BACK =       23,
    VERB_K_REF  =       24,
    VERB_K_TOP  =       25,
    VERB_K_HELP =       26,
    VERB_K_ASC  =       27,
    VERB_K_QUIT =       28,
    VERB_K_SHL  =       29,
    VERB_K_SHR  =       30,
    VERB_K_TAB  =       31,
    VERB_K_TC   =       32,
    VERB_K_TD   =       33,
    VERB_K_TI   =       34,
    VERB_K_EXT  =       35,
    VERB_K_KS   =       36,
    VERB_K_DEFK =       37,
    VERB_K_BELL =       38,
    VERB_K_DATE =       39,
    VERB_K_DUPC =       40,
    VERB_K_DLWC =       41,
    VERB_K_DMOV =       42,
    VERB_K_DESEL =      43,
    VERB_K_TGSEL =      44,
    VERB_K_CLSS =       45,
    LAST_K_VERB =       45;

!
! Changecase types.
!

LITERAL
        CASE_K_CHGC =   1,      ! Invert case, corresponds to VERB_K_CHGC
        CASE_K_CHGU =   2,      ! Upper case,  corresponds to VERB_K_CHGU
        CASE_K_CHGL =   3;      ! Lower case,  corresponds to VERB_K_CHGL
```

```
!+
! PARSER OP-CODE DEFINITIONS
!
! The following are the op-codes accepted by the parser driver.
!-

LITERAL
        OPC_ABORT       =       0,      ! Abort the parse
        OPC_ACTION      =       1,      ! Perform action routine
        OPC_CALL        =       2,      ! Call sub-table
        OPC_RETURN      =       3,      ! End of table or sub-table (return)
        OPC_GOTO        =       4,      ! Unconditional goto
        OPC_OPTION      =       5,      ! Optional phrase check
        OPC_REQUIRE     =       6,      ! Require a specific token
        OPC_SELECT      =       7,      ! select one of several options

        OP_ABORT        =       0,      ! now the bit values
        OP_ACTION       =       32,
        OP_CALL         =       64,
        OP_RETURN       =       96,
        OP_GOTO         =       128,
        OP_OPTION       =       160,
        OP_REQUIRE      =       192,
        OP_SELECT       =       224;

!+
! Token class definitions
!-

LITERAL
        CL_NAME         =       0,      ! name class
        CL_NUMBER       =       1,      ! the number class
        CL_SPECIAL      =       2,      ! the special character class
        CL_STRING       =       3;      ! The qouted string class

!+
! Parser token handling and matching macros
!-

MACRO
        PAR_MIN_LENGTH = 0,0,3,0 %,
        PAR_MAX_LENGTH = 0,4,4,0 %,
        PAR_OPT_PERCENT = 0,3,1,0 %,
        PAR_SYMBOL = 1,0,0,0 %;
```

```
!+
! Miscellaneous definitions
!-


%IF %BLISS(BLISS32) %THEN

MACRO STRING_DESC(DESC,LEN,ADDR) =
BEGIN EXTERNAL ROUTINE STR$COPY_R; STR$COPY_R(DESC,LEN,ADDR) END %;

%ELSE
!+
! These DSC$ macros are defined as system symbols on VAX/VMS.  They are
! fields in a string descriptor.  To get the effect of a string descriptor
! on the 11's, we will pass a 4 word field with the following macros defining
! the pointer to the string address and the field of the string length.
!-


MACRO
    DSC$A_POINTER = 4,0,16,0%,
    DSC$W_LENGTH = 0,0,16,0%;

MACRO STRING_DESC ( DESC, LEN, ADDR) =
    BEGIN
    MAP
        DESC: BLOCK[8,BYTE];
    DESC[DSC$A_POINTER] = ADDR;
    DESC[DSC$W_LENGTH] = .LEN;
    END %;

%FI

LITERAL
    NO_UPDATE   =       256,    ! Indicating no update of current line needed
    NO_REFRESH  =       100,    ! Indicating no refresh of screen needed
    MESSAGE_LINE=       22,     ! Line on which messages are displayed
    COMMAND_LINE=       23,     ! Line on which command prompts are displayed
    DIR_FORWARD =       1,      ! Forward direction.
    DIR_BACKWARD=       0;      ! Backward direction.

!+
! Definition of the ASSERT macro.  This macro calls EDT$$INTER_ERR if the
! condition is not true.
!-

MACRO ASSERT (CONDITION) =
    BEGIN
    IF (NOT (CONDITION))
    THEN
        BEGIN
        EXTERNAL ROUTINE EDT$$INTER_ERR : NOVALUE;
        EDT$$INTER_ERR ();
        END;
    END
```

```
 %;
```

```
!+
! Symbols used in control C journaling.
!-
LITERAL
        CC_REC_SIZE = 6,                ! Size of a control C record
        JOU_REC_ESC = %X'FF',           ! First (escape) byte of a non-text record in the journal file
        CC_REC_FLAG = 1,                ! Second byte: control C record
        CC_CTR_MAX = 30000;             ! Maximum counter value in control C handling

!+
! Symbol used in the formatter
!-
%IF %BLISS(BLISS32) %THEN
LITERAL
        EDT$$K_FMT_BUFLEN = 512;        ! Length of the format buffer

%ELSE
LITERAL
        EDT$$K_FMT_BUFLEN = 136;        ! Length of the format buffer

%FI

!       End of file EDT.REQ
```

0129 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

EDT

EDT
MAP

EDFVAR
LIS

EDTSHR
MAP

EDT
REQ

0130 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

ERRMSG
REQ

TRACEOFF
REQ

SUPPORTS
REQ

TRAROUNAM
REQ

BADKEY
LIS

CHMCMBLIN
LIS

TRACEON
REQ

TRANSLATE
REQ

CALLWIO
LIS

CHMBEGSEN
LIS

CHMBELL
LIS

SYSSYM
REQ

TRANNAMES
REQ

CHMCHGC
LIS

CHMDELLIN
LIS

TRACELIT
REQ

EDTREQ
REQ

KEYPADDEF
REQ

CALLFIO
LIS

CHMBEGWRD
LIS

CHMCLRCC
LIS

CHMBEEP
LIS

CHMCHANGE
LIS

TRACEMAC
REQ

CHMDELCHR
LIS

PSECTS
REQ

VERSION
REQ

CHMCHKCC
LIS

CHMEINPUT
LIS