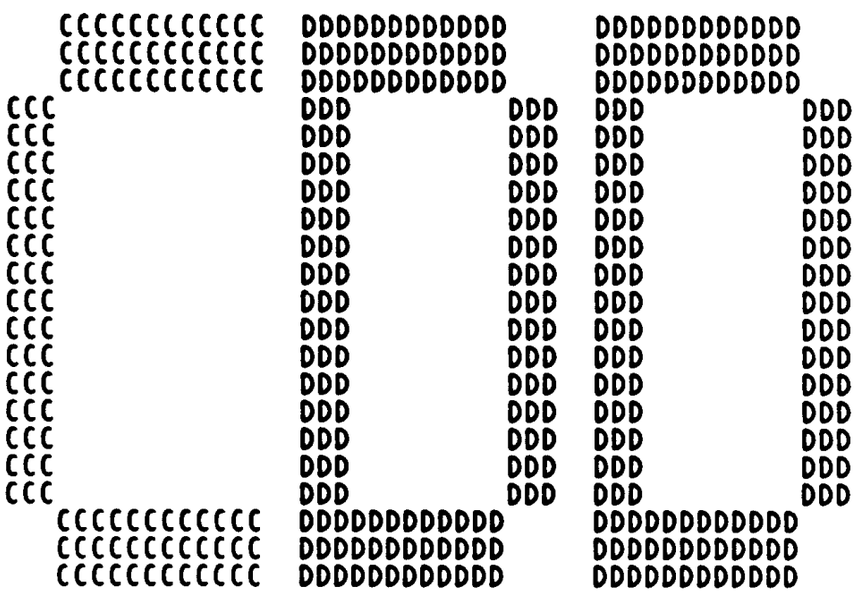
CCCCCCCCCCC DDDDDDDDDDD DDDDDDDDDDD
CCCCCCCCCCC DDDDDDDDDDD DDDDDDDDDDD
CCCCCCCCCCC DDDDDDDDDDD DDDDDDDDDDD
CCC DDD DDD DDD DDD
CCC DDD DDD DDD DDD
CCC DDD DDD DDD DDD
CCC DDD DDD DDD DDD
CCC DDD DDD DDD DDD
CCC DDD DDD DDD DDD
CCC DDD DDD DDD DDD
CCC DDD DDD DDD DDD
CCC DDD DDD DDD DDD
CCC DDD DDD DDD DDD
CCC DDD DDD DDD DDD
CCC DDD DDD DDD DDD
CCC DDD DDD DDD DDD
CCC DDD DDD DDD DDD
CCCCCCCCCCC DDDDDDDDDDD DDDDDDDDDDD
CCCCCCCCCCC DDDDDDDDDDD DDDDDDDDDDD
CCCCCCCCCCC DDDDDDDDDDD DDDDDDDDDDD

```
  CCCCCCCC  DDDDDDDD    DDDDDDDD    LL          IIIIII    BBBBBBBB
  CCCCCCCC  DDDDDDDD    DDDDDDDD    LL          IIIIII    BBBBBBBB
CC          DD      DD  DD      DD  LL            II      BB      BB
CC          DD      DD  DD      DD  LL            II      BB      BB
CC          DD      DD  DD      DD  LL            II      BB      BB
CC          DD      DD  DD      DD  LL            II      BB      BB
CC          DD      DD  DD      DD  LL            II      BBBBBBBB
CC          DD      DD  DD      DD  LL            II      BBBBBBBB
CC          DD      DD  DD      DD  LL            II      BB      BB
CC          DD      DD  DD      DD  LL            II      BB      BB
CC          DD      DD  DD      DD  LL            II      BB      BB      ....
CC          DD      DD  DD      DD  LL            II      BB      BB      ....
  CCCCCCCC  DDDDDDDD    DDDDDDDD    LLLLLLLLLL  IIIIII    BBBBBBBB        ....
  CCCCCCCC  DDDDDDDD    DDDDDDDD    LLLLLLLLLL  IIIIII    BBBBBBBB        ....
```

```
LL            IIIIII      SSSSSSSS
LL            IIIIII      SSSSSSSS
LL              II      SS
LL              II      SS
LL              II      SS
LL              II      SS
LL              II        SSSSSS
LL              II        SSSSSS
LL              II              SS
LL              II              SS
LL              II              SS
LL              II              SS
LLLLLLLLLL    IIIIII    SSSSSSSS
LLLLLLLLLL    IIIIII    SSSSSSSS
```

```
0001  0  !*****************************************************************************
0002  0  !*                                                                           *
0003  0  !*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                  *
0004  0  !*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                   *
0005  0  !*  ALL RIGHTS RESERVED.                                                     *
0006  0  !*                                                                           *
0007  0  !*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED     *
0008  0  !*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE    *
0009  0  !*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER     *
0010  0  !*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY     *
0011  0  !*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY     *
0012  0  !*  TRANSFERRED.                                                             *
0013  0  !*                                                                           *
0014  0  !*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE     *
0015  0  !*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT    *
0016  0  !*  CORPORATION.                                                             *
0017  0  !*                                                                           *
0018  0  !*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS     *
0019  0  !*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                  *
0020  0  !*                                                                           *
0021  0  !*                                                                           *
0022  0  !*****************************************************************************
0023  0
0024  0  !++
0025  0  !
0026  0  !  TITLE:  CDDLIB                          CDD Bliss Library
0027  0  !
0028  0  !  VERSION:  'V04-000'
0029  0  !
0030  0  !  FACILITY:  Common Data Dictionary
0031  0  !
0032  0  !  ABSTRACT:
0033  0  !
0034  0  !       This module is the library file used for compiling all other
0035  0  !       modules in the CDD facility.
0036  0  !
0037  0  !  ENVIRONMENT:
0038  0  !
0039  0  !  AUTHOR:  Jeff East,  22-Jan-80
0040  0  !
0041  0  !  MODIFIED BY:
0042  0  !
0043  0  !       7-May-81 (JAE)  Added $IO_SYNC macro.
0044  0  !
0045  0  !--
0046  0      %TITLE      'CDD Bliss Library'
```

```
;  0047  0          %SBTTL      'STRUCTURE DEFINITIONS'
;  0048  0      !++
;  0049  0      !
;  0050  0      !       STRUCTURE DEFINITIONS
;  0051  0      !
;  0052  0      !--
;  0053  0      !
;  0054  0      !
;  0055  0      !+
;  0056  0      !       On-disk Pointer Structure
;  0057  0      !
;  0058  0      !
;  0059  0      !               23                  6      0
;  0060  0      !                +------------------+------+ +0
;  0061  0      !                |                  |      |
;  0062  0      !                |  Logical Page    | Line |      DPTR$V_LINE
;  0063  0      !                |     number       |number|      DPTR$V_PAGE
;  0064  0      !                +------------------+------+ +3
;  0065  0      !
;  0066  0      !-
;  0067  0
;  0068  0      LITERAL
;  0069  0          DPTR$S_BLOCK_LENGTH = 3;
;  0070  0
;  0071  0      MACRO
;M 0072  0          $DPTR = BLOCK[DPTR$S_BLOCK_LENGTH,BYTE] FIELD (DPTR$Z_FIELDS)
;  0073  0          %;
;  0074  0
;  0075  0      FIELD  DPTR$Z_FIELDS =
;  0076  0          SET
;  0077  0              DPTR$V_VALUE            = [0, 0, 24, 0],
;  0078  0              DPTR$V_LINE             = [0, 0, 7, 0],
;  0079  0              DPTR$V_PAGE             = [0, 7, 17, 0]
;  0080  0          TES;
```

```
;    0081  0     !+
;    0082  0     !       In-core Disk Pointer Structure
;    0083  0     !
;    0084  0     !
;    0085  0     !      31       23                  6      0
;    0086  0     !       +--------+-----------------+------+ +0
;    0087  0     !       |Internal|                 |      |      DKEY$V_LINE
;    0088  0     !       |  file  |  Logical Page   | Line |      DKEY$V_PAGE
;    0089  0     !       | number |     number      |number|      DKEY$B_FILE
;    0090  0     !       +--------+-----------------+------+ +4
;    0091  0     !
;    0092  0     !       When a disk pointer is being passed around routines, it
;    0093  0     !       is passed as a Disk Key (DKEY), rather than just a disk
;    0094  0     !       pointer.
;    0095  0     !
;    0096  0     !       Disk Keys include all the information of a disk pointer,
;    0097  0     !       but also include a pointer to their file's FCB.
;    0098  0     !-
;    0099  0
;    0100  0     LITERAL
;    0101  0         DKEY$S_BLOCK_LENGTH = 4;
;    0102  0
;    0103  0     MACRO
; M  0104  0         $DKEY = BLOCK[DKEY$S_BLOCK_LENGTH,BYTE] FIELD (DKEY$Z_FIELDS)
;    0105  0         %;
;    0106  0
;    0107  0     FIELD  DKEY$Z_FIELDS =
;    0108  0         SET
;    0109  0             DKEY$V_LINE             = [0, 0, 7, 0],
;    0110  0             DKEY$V_PAGE             = [0, 7, 17, 0],
;    0111  0             DKEY$B_FILE             = [0, 24, 8, 0]
;    0112  0         TES;
```

```
;    0113  0      !+
;    0114  0      !       In-core Page Number Structure
;    0115  0      !
;    0116  0      !
;    0117  0      !      31       23                       0
;    0118  0      !       +--------+-----------------------+ +0
;    0119  0      !       |        |                       |
;    0120  0      !       |  File  |      Page number      |      PKEY$V_PAGE
;    0121  0      !       | number |                       |      PKEY$B_FILE
;    0122  0      !       +--------+-----------------------+ +4
;    0123  0      !
;    0124  0      !       Dictionary page numbers that are passed around the hash table
;    0125  0      !       use the Page Key (PKEY) structure to provide both the page
;    0126  0      !       and file numbers.
;    0127  0      !-
;    0128  0
;    0129  0      LITERAL
;    0130  0          PKEY$S_BLOCK_LENGTH = 4;
;    0131  0
;    0132  0      MACRO
;M   0133  0          $PKEY = BLOCK[PKEY$S_BLOCK_LENGTH,BYTE] FIELD (PKEY$Z_FIELDS)
;    0134  0          %;
;    0135  0
;    0136  0      FIELD  PKEY$Z_FIELDS =
;    0137  0          SET
;    0138  0              PKEY$V_PAGE             = [0, 0, 24, 0],
;    0139  0              PKEY$B_FILE             = [0, 24, 8, 0]
;    0140  0          TES;
```

```
;  0141  0     !+
;  0142  0     !       Line Index Format
;  0143  0     !
;  0144  0     !
;  0145  0     !        31              15              0
;  0146  0     !                        +---------------+ +0
;  0147  0     !                        |               |
;  0148  0     !                        | Offset to blk |      LINE$W_BLOCK_OFFSET
;  0149  0     !                        |from page base |
;  0150  0     !                        +---------------+ +2
;  0151  0     !
;  0152  0     !       Each block on a dictionary page is pointed to by a line index
;  0153  0     !       on that page.  Disk pointers (DPTR) between the various blocks
;  0154  0     !       actually point to the block's line index.  The line index is
;  0155  0     !       then used to locate the physical dictionary block.
;  0156  0     !
;  0157  0     !       The PAGE$INDEX macro allows the programmer to access a line
;  0158  0     !       index while it resides on a dictionary page.  It also uses the
;  0159  0     !       LINE$Z_FIELDS to access the individual fields within a line index.
;  0160  0     !-
;  0161  0
;  0162  0     LITERAL
;  0163  0         LINE$S_BLOCK_LENGTH = 2;
;  0164  0
;  0165  0     MACRO
;M 0166  0         $LINE = BLOCK[LINE$S_BLOCK_LENGTH,BYTE] FIELD (LINE$Z_FIELDS)
;  0167  0         %;
;  0168  0
;  0169  0     FIELD  LINE$Z_FIELDS =
;  0170  0         SET
;  0171  0             LINE$W_BLOCK_OFFSET     = [0, 0, 16, 0]
;  0172  0         TES;
```

```
;  0173  0     !+
;  0174  0     !       Page Format
;  0175  0     !
;  0176  0     !
;  0177  0     !        31                15                0
;  0178  0     !       +---------------------------------+ +0
;  0179  0     !       |                                 |
;  0180  0     !       |          Page Checksum          |      PAGE$L_CHECKSUM
;  0181  0     !       |                                 |
;  0182  0     !       +-------+------+------------------+ +4
;  0183  0     !       |Page's |      |                  |
;  0184  0     !       |cluster|      |   Page Number    |      PAGE$V_NUMBER
;  0185  0     !       | seq # |      |                  |      PAGE$B_CLUSTER_IO_SEQ
;  0186  0     !       +-------+------+------------------+ +8
;  0187  0     !       |                                 |      PAGE$L_NEXT_PAGE
;  0188  0     !       | Size of next | Next Related Page|      PAGE$V_NEXT_PAGE
;  0189  0     !       |  page group  |                  |      PAGE$V_NEXT_SIZE
;  0190  0     !       +-------+------+------------------+ +12
;  0191  0     !       |Page's |      |      Prior       |
;  0192  0     !       |group  |      | Related Page     |      PAGE$V_PRIOR_PAGE
;  0193  0     !       |seq #  |      |                  |      PAGE$B_GROUP_IO_SEQ
;  0194  0     !       +-------+------+--+---------------+ +16
;  0195  0     !       |                 |               |
;  0196  0     !       |  Number of      |  Free space   |      PAGE$W_FREE_SPACE
;  0197  0     !       |  line indices   |               |      PAGE$W_INDICES
;  0198  0     !       +-----------------+---------------+ +20  PAGE$V_DATA
;  0199  0     !       |                                 |
;  0200  0     !       |           Data area             |
;  0201  0     !       |                                 |
;  0202  0     !       +/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/+
;  0203  0     !       |                                 |
;  0204  0     !       |        Line index area          |
;  0205  0     !       |                                 |
;  0206  0     !       +---------------------------------+ +512  PAGE$V_INDEX_BASE
;  0207  0     !
;  0208  0     !
;  0209  0     !       Each page in the dictionary file starts with a page header.
;  0210  0     !       The next and prior related page pointers are used for linking
;  0211  0     !       page groups.  The following types of page groups exist:
;  0212  0     !
;  0213  0     !               1)  Lockable page group.
;  0214  0     !                   Each named entity (and history list) owns exactly
;  0215  0     !                   one lockable page group.  These groups contain
;  0216  0     !                   all unamed offspring of the name entiry (or history
;  0217  0     !                   list).
;  0218  0     !
;  0219  0     !                   A page group may only be accessed through its
;  0220  0     !                   portal page.  A group's portal page is the page
;  0221  0     !                   on which the named entity (or history list head)
;  0222  0     !                   resides.  If a group's portal page is locked, then
;  0223  0     !                   no pages in the group may be accessed.
;  0224  0     !
;  0225  0     !       The PAGE$INDEX structure is used to access a line index entry on
;  0226  0     !       a page.
;  0227  0     !
;  0228  0     !       A page whose checksum is zero is a locked page, and indicates that
;  0229  0     !       the sub-tree below it is incomplete and in a transient state.
```

```
;    0230  0      !       Such pages may never be read.
;    0231  0      !
;    0232  0      !       Each page has two page sequence numbers.  The cluster sequence
;    0233  0      !       number must be the same for all pages in the cluster.  The group
;    0234  0      !       sequence number must be the same for all pages in an I/O group.
;    0235  0      !
;    0236  0      !       The cluster I/O sequence number is bumped whenever more than one
;    0237  0      !       group in the cluster is written.  The group I/O sequence number
;    0238  0      !       is bumped whenever the group is written.  These sequence numbers
;    0239  0      !       enable the detection of incomplete cluster unstage operations.
;    0240  0      !
;    0241  0      !-
;    0242  0
;    0243  0      LITERAL
;    0244  0          PAGE$S_BLOCK_LENGTH = 512;
;    0245  0
;    0246  0      MACRO
;M   0247  0          $PAGE = BLOCK[PAGE$S_BLOCK_LENGTH,BYTE] FIELD (PAGE$Z_FIELDS)
;    0248  0          %;
;    0249  0
;    0250  0      FIELD PAGE$Z_FIELDS =
;    0251  0          SET
;    0252  0              PAGE$L_CHECKSUM         = [0, 0, 32, 0],
;    0253  0              PAGE$V_NUMBER           = [4, 0, 17, 0],
;    0254  0              PAGE$B_CLUSTER_IO_SEQ   = [7, 0, 8, 0],
;    0255  0              PAGE$L_NEXT_PAGE        = [8, 0, 32, 0],
;    0256  0              PAGE$V_NEXT_PAGE        = [8, 0, 17, 0],
;    0257  0              PAGE$V_NEXT_SIZE        = [8, 17, 15, 0],
;    0258  0              PAGE$V_PRIOR_PAGE       = [12, 0, 17, 0],
;    0259  0              PAGE$B_GROUP_IO_SEQ     = [15, 0, 8, 0],
;    0260  0              PAGE$W_FREE_SPACE       = [16, 0, 16, 0],
;    0261  0              PAGE$W_INDICES          = [18, 0, 16, 0],
;    0262  0              PAGE$V_DATA             = [20, 0, 0, 0],
;    0263  0              PAGE$V_INDEX_BASE       = [512, 0, 0, 0]
;    0264  0          TES;
;    0265  0
;    0266  0      LITERAL
;    0267  0          PAGE$K_BASE         = BLOCK[0, PAGE$V_DATA;
;    0268  0                                     PAGE$S_BLOCK_LENGTH, BYTE],
;    0269  0          PAGE$K_FREE_SPACE   = PAGE$S_BLOCK_LENGTH - PAGE$K_BASE,
;    0270  0          PAGE$S_INDEX_BASE   = BLOCK[0, PAGE$V_INDEX_BASE;
;    0271  0                                     PAGE$S_BLOCK_LENGTH, BYTE];
;    0272  0
;    0273  0      STRUCTURE
;    0274  0          PAGE$INDEX[I, O, P, S, E] =
;    0275  0              (PAGE$INDEX+PAGE$S_INDEX_BASE+O-((I)*LINE$S_BLOCK_LENGTH)) <P,S,E>;
```

```
;    0276  0      !+
;    0277  0      !       Dictionary Header Block
;    0278  0      !
;    0279  0      !
;    0280  0      !               23        15              0
;    0281  0      !                +-----------------------+ +0
;    0282  0      !                |                       |
;    0283  0      !                |  Pointer to top of    |      DICT$V_CDD$TOP
;    0284  0      !                |  dictionary sub-tree  |
;    0285  0      !                +-----------------------+ +3
;    0286  0      !                |                       |
;    0287  0      !                |  Pointer to last      |      DICT$V_LAST_BITMAP
;    0288  0      !                |  free page bit-map    |
;    0289  0      !                +-----------------------+ +6
;    0290  0      !
;    0291  0      !
;    0292  0      !       Each dictionary uses page 1 as its dictionary header page.
;    0293  0      !       The dictionary header block immediately follows the page
;    0294  0      !       header on page 1.
;    0295  0      !
;    0296  0      !       DICT$V_CDD$TOP points to the root of the tree/sub-tree contained
;    0297  0      !       in the dictionary file.
;    0298  0      !
;    0299  0      !       DICT$V_LAST_BITMAP points to the last free page bit-map entry.
;    0300  0      !-
;    0301  0
;    0302  0      LITERAL
;    0303  0          DICT$S_BLOCK_LENGTH = 6 + PAGE$K_BASE;
;    0304  0
;    0305  0      MACRO
; M  0306  0          $DICT = BLOCK[DICT$S_BLOCK_LENGTH,BYTE] FIELD (PAGE$Z_FIELDS,
; M  0307  0                                                         DICT$Z_FIELDS)
;    0308  0          %;
;    0309  0
;    0310  0      FIELD DICT$Z_FIELDS =
;    0311  0          SET
;    0312  0              DICT$V_CDD$TOP          = [0+PAGE$K_BASE, 0, 24, 0],
;    0313  0              DICT$V_LAST_BITMAP      = [3+PAGE$K_BASE, 0, 24, 0]
;    0314  0          TES;
```

```
;  0315  0     !+
;  0316  0     !       Universal Block Header
;  0317  0     !
;  0318  0     !      31       24      15              0
;  0319  0     !       +----------------+-------+-------+ +0
;  0320  0     !       |                |       |       |      BLK$B_TYPE
;  0321  0     !       |  Block Length  | Block | Block |      BLK$B_FLAGS
;  0322  0     !       |                | Flags | Type  |      BLK$W_LENGTH
;  0323  0     !       +----------------+-------+-------+ +4
;  0324  0     !
;  0325  0     !
;  0326  0     !       This block header is used in every non-fixed block on the
;  0327  0     !       dictionary pages.  Fixed blocks (one which always occur
;  0328  0     !       in the same place whenever they exist) do not have this
;  0329  0     !       block header.
;  0330  0     !
;  0331  0     !-
;  0332  0
;  0333  0     LITERAL
;  0334  0         BLK$S_BLOCK_LENGTH  = 4;
;  0335  0
;  0336  0     MACRO
;M 0337  0         $BLK = BLOCK[BLK$S_BLOCK_LENGTH,BYTE] FIELD (ACL$Z_FIELDS,
;M 0338  0                                                      ACLC$Z_FIELDS,
;M 0339  0                                                      ATT$Z_FIELDS,
;M 0340  0                                                      BLK$Z_FIELDS,
;M 0341  0                                                      ELST$Z_FIELDS,
;M 0342  0                                                      ENT$Z_FIELDS,
;M 0343  0                                                      FPCB$Z_FIELDS,
;M 0344  0                                                      LIST$Z_FIELDS,
;M 0345  0                                                      LST$Z_FIELDS,
;M 0346  0                                                      NAME$Z_FIELDS,
;M 0347  0                                                      NAT$Z_FIELDS,
;M 0348  0                                                      NNAM$Z_FIELDS,
;M 0349  0                                                      NODE$Z_FIELDS,
;M 0350  0                                                      SEG$Z_FIELDS,
;M 0351  0                                                      SLST$Z_FIELDS,
;M 0352  0                                                      SSA$Z_FIELDS,
;M 0353  0                                                      STR$Z_FIELDS,
;M 0354  0                                                      TEXT$Z_FIELDS)
;  0355  0             %;
;  0356  0
;  0357  0
;  0358  0     FIELD BLK$Z_FIELDS =
;  0359  0         SET
;  0360  0             BLK$B_TYPE              = [0, 0, 8, 0],
;  0361  0             BLK$B_FLAGS             = [1, 0, 8, 0],
;  0362  0             BLK$W_LENGTH            = [2, 0, 16, 0]
;  0363  0         TES;
;  0364  0
;  0365  0     LITERAL
;  0366  0         BLK$K_FLAGS         = BLOCK[0, BLK$B_TYPE; ,BYTE],
;  0367  0         BLK$K_BASE          = BLK$S_BLOCK_LENGTH;
;  0368  0
;  0369  0
;  0370  0     !+
;  0371  0     !       Block types
```

```
;    0372  0      !
;    0373  0      !       The ordering and clustering of these block types is significant.
;    0374  0      !
;    0375  0      !       All block types must be contiguous without any holes and
;    0376  0      !       bounded by the symbols BLK$K_TYPE_FIRST and BLK$K_TYPE_LAST.
;    0377  0      !
;    0378  0      !       All node and NAME block types must be contiguous and bounded
;    0379  0      !       by the symbols BLK$K_TYPE_NODE_FIRST and BLK$K_TYPE_NODE_LAST.
;    0380  0      !
;    0381  0      !       To add more node or NAME block types, insert them at the front
;    0382  0      !       of the table.
;    0383  0      !
;    0384  0      !       To add any other block type, append them to the end of the table.
;    0385  0      !
;    0386  0      !       CHANGING THE VALUES OF ANY OF THE ACTUAL BLOCK TYPES WILL INVALIDATE
;    0387  0      !       EXISTING DICTIONARIES.
;    0388  0      !-
;    0389  0
;    0390  0      LITERAL
;    0391  0          BLK$K_TYPE_FIRST                    = 101,  ! Lowest block type
;    0392  0
;    0393  0              BLK$K_TYPE_NODE_FIRST           = 101,  ! First node or NAM block
;    0394  0
;    0395  0                  BLK$K_TYPE_DIR_NAM          = 101,  ! Directory NAM block
;    0396  0                  BLK$K_TYPE_FIL_NAM          = 102,  ! File NAM block
;    0397  0                  BLK$K_TYPE_TERM_NAM         = 103,  ! Terminal NAM block
;    0398  0                  BLK$K_TYPE_DIR_NODE         = 104,  ! Directory node block
;    0399  0                  BLK$K_TYPE_TERM_NODE        = 105,  ! Terminal node block
;    0400  0
;    0401  0              BLK$K_TYPE_NODE_LAST            = 105,  ! Last node or NAM block
;    0402  0
;    0403  0              BLK$K_TYPE_BITMAP               = 106,  ! Free page bitmap
;    0404  0              BLK$K_TYPE_ENTITY_ATT           = 107,  ! Entity attribute block
;    0405  0              BLK$K_TYPE_ENTITY_LIST          = 108,  ! Entity list block
;    0406  0              BLK$K_TYPE_ENTITY_LIST_ATT      = 109,  ! Entity list attribute block
;    0407  0              BLK$K_TYPE_NULL_ATT             = 110,  ! Null attribute block
;    0408  0              BLK$K_TYPE_NUM_ATT              = 111,  ! Numeric attribute block
;    0409  0              BLK$K_TYPE_SHORT_ATT            = 112,  ! Short string attribute block
;    0410  0              BLK$K_TYPE_STRING_ATT           = 113,  ! String attribute block
;    0411  0              BLK$K_TYPE_STRING_LIST          = 115,  ! String list block
;    0412  0              BLK$K_TYPE_STRING_LIST_ATT      = 116,  ! String list attribute block
;    0413  0              BLK$K_TYPE_STRING_SEG           = 117,  ! String segment block
;    0414  0              BLK$K_TYPE_TEXT                 = 118,  ! Text block
;    0415  0              BLK$K_TYPE_ACL                  = 119,  ! Access control list entry
;    0416  0              BLK$K_TYPE_ACLC                 = 120,  ! Access control list criterion
;    0417  0          BLK$K_TYPE_LAST                     = 120;  ! Highest block type
```

```
;  0418  0     !+
;  0419  0     !       Free Page Control Block (FPCB)
;  0420  0     !       Free Page Bit Map (FPBM)
;  0421  0     !
;  0422  0     !
;  0423  0     !               +-----------------------+ +0
;  0424  0     !               |                       |
;  0425  0     !               | Disk address of prior |       FPCB$V_PRIOR
;  0426  0     !               |        FBCB           |
;  0427  0     !       +-------+-------+---------------+ +3
;  0428  0     !       |               |               |
;  0429  0     !       |  Number of    |  First page   |       FPCB$V_FIRST_PAGE
;  0430  0     !       | pages mapped  |    mapped     |       FPCB$V_PAGE_COUNT
;  0431  0     !       +---------------+---------------+ +7
;  0432  0     !                       |               |
;  0433  0     !                       | Blocks not    |       FPCB$W_EXCESS_BLOCKS
;  0434  0     !                       | used here     |
;  0435  0     !                       +---------------+ +9    FPCB$V_BITMAP
;  0436  0     !
;  0437  0     !       Free pages in the dictionary file are controlled by the Free Page
;  0438  0     !       Control Blocks (FPCB) and the Free Page Bit Map (FPBM).  Each
;  0439  0     !       free page in the file has its corresponding bit set on.
;  0440  0     !
;  0441  0     !       The disk's cluster size may make it impossible to use all of the
;  0442  0     !       allocated blocks.  If so, the un-used blocks are tallied in
;  0443  0     !       FPCB$W_EXCESS_BLOCKS, and will be used when the file is next
;  0444  0     !       extended.
;  0445  0     !
;  0446  0     !       The FPCBs are chained as the dictionary file grows in size.
;  0447  0     !
;  0448  0     !-
;  0449  0
;  0450  0     LITERAL
;  0451  0         FPCB$S_BLOCK_LENGTH = BLK$K_BASE + 9;
;  0452  0
;  0453  0     STRUCTURE
;  0454  0         FPCB[O, P, S, E; BLOCKS] = [FPCB$S_BLOCK_LENGTH+(BLOCKS+7)/8]
;  0455  0                                    (FPCB+O)<P,S,E>;
;  0456  0
;  0457  0     FIELD  FPCB$Z_FIELDS =
;  0458  0         SET
;  0459  0             FPCB$V_PRIOR            = [0+BLK$K_BASE, 0, 24, 0],
;  0460  0             FPCB$V_FIRST_PAGE       = [3+BLK$K_BASE, 0, 17, 0],
;  0461  0             FPCB$V_PAGE_COUNT       = [3+BLK$K_BASE, 17, 15, 0],
;  0462  0             FPCB$W_EXCESS_BLOCKS    = [7+BLK$K_BASE, 0, 16, 0],
;  0463  0             FPCB$V_BITMAP           = [9+BLK$K_BASE, 0, 0, 0]
;  0464  0         TES;
;  0465  0
;  0466  0     LITERAL
;  0467  0         FPCB$K_BASE   = FPCB$S_BLOCK_LENGTH,
;  0468  0         FPCB$S_BITMAP = FPCB[0, FPCB$V_BITMAP];
;  0469  0
;  0470  0     STRUCTURE
;  0471  0         FPBM[I] = (FPBM+FPCB$S_BITMAP)<I-1, 1, 0>;
```

```
;  0472  0      !+
;  0473  0      !       Name Block
;  0474  0      !
;  0475  0      !
;  0476  0      !        31      24        15                0
;  0477  0      !                +------------------------+ +0
;  0478  0      !                |                        |
;  0479  0      !                |  Pointer to next NAME  |      NAME$V_SIBLING
;  0480  0      !                |       (sibling)        |
;  0481  0      !                +------------------------+ +3
;  0482  0      !                |                        |
;  0483  0      !                |  Pointer to name block |      NAME$V_NAME
;  0484  0      !                |                        |
;  0485  0      !                +-------+----------------+ +6
;  0486  0      !                |   Pointer to node      |
;  0487  0      !                |  block or forwarding   |      NAME$V_NODE or NAME$V_FILE
;  0488  0      !                |       file name        |
;  0489  0      !                +------------------------+ +9
;  0490  0      !
;  0491  0      !       There are three types of NAM Blocks, Directory, File, and Terminal.
;  0492  0      !
;  0493  0      !       Directory NAM Block is made up of a Block Header + NAM Block.
;  0494  0      !
;  0495  0      !       File NAM Block is made up of a Block Header + NAM Block.
;  0496  0      !
;  0497  0      !       Terminal NAM Block is made up of a Block Header + NAM Block + Terminal
;  0498  0      !       NAM Block.
;  0499  0      !
;  0500  0      !-
;  0501  0
;  0502  0      LITERAL
;  0503  0          NAME$S_BLOCK_LENGTH = 9 + BLK$K_BASE;
;  0504  0
;  0505  0      MACRO
;M 0506  0          $NAME = BLOCK[NAME$S_BLOCK_LENGTH,BYTE] FIELD (BLK$Z_FIELDS,
;M 0507  0                                                         NAME$Z_FIELDS)
;  0508  0          %;
;  0509  0
;  0510  0      FIELD NAME$Z_FIELDS =
;  0511  0          SET
;  0512  0              NAME$V_SIBLING          = [0+BLK$K_BASE, 0, 24, 0],
;  0513  0              NAME$V_NAME             = [3+BLK$K_BASE, 0, 24, 0],
;  0514  0              NAME$V_NODE             = [6+BLK$K_BASE, 0, 24, 0],
;  0515  0              NAME$V_FILE             = [6+BLK$K_BASE, 0, 24, 0]
;  0516  0          TES;
;  0517  0
;  0518  0      LITERAL
;  0519  0          NAME$K_BASE          = NAME$S_BLOCK_LENGTH;
```

```
;    0520  0      !+
;    0521  0      !       Node Nam Block
;    0522  0      !
;    0523  0      !
;    0524  0      !        31      24         15                0
;    0525  0      !         +-----------------------+ +0
;    0526  0      !         |                       |
;    0527  0      !         |  Pointer to resource  |       NNAM$V_RESOURCE
;    0528  0      !         |       name block      |
;    0529  0      !         +-----------------------+ +3
;    0530  0      !
;    0531  0      !       Node NAM Block is made up of a Block Header + NAM Block + Node
;    0532  0      !       NAM Block.
;    0533  0      !
;    0534  0      !-
;    0535  0
;    0536  0      LITERAL
;    0537  0          NNAM$S_BLOCK_LENGTH = 3 + NAME$K_BASE;
;    0538  0
;    0539  0      MACRO
;  M 0540  0          $NNAM = BLOCK[NNAM$S_BLOCK_LENGTH,BYTE] FIELD (BLK$Z_FIELDS,
;  M 0541  0                                                         NAME$Z_FIELDS,
;  M 0542  0                                                         NNAM$Z_FIELDS)
;    0543  0          %;
;    0544  0
;    0545  0      FIELD NNAM$Z_FIELDS =
;    0546  0          SET
;    0547  0              NNAM$V_RESOURCE = [0+NAME$K_BASE, 0, 24, 0]
;    0548  0          TES;
;    0549  0
;    0550  0      LITERAL
;    0551  0          NNAM$K_BASE          = NNAM$S_BLOCK_LENGTH;
```

```
;  0552  0     !+
;  0553  0     !       Node Block
;  0554  0     !
;  0555  0     !
;  0556  0     !        31      24        15              0
;  0557  0     !         +-----------------------------+ +0
;  0558  0     !         |                             |
;  0559  0     !         |  Pointer to name block      |      NODE$V_NAME
;  0560  0     !         |                             |
;  0561  0     !         +-----------------------------+ +3
;  0562  0     !         |                             |
;  0563  0     !         |  Pointer to history         |      NODE$V_HISTORY
;  0564  0     !         |       list head             |
;  0565  0     !         +-----------------------------+ +6
;  0566  0     !         |                             |
;  0567  0     !         |  Pointer to protection      |      NODE$V_PROTECTION
;  0568  0     !         |       list head             |
;  0569  0     !         +-------+---------------------+ +9
;  0570  0     !         |                             |
;  0571  0     !         |    Pointer to parent        |      NODE$V_PARENT
;  0572  0     !         |         node                |
;  0573  0     !         +-----------------------------+ +12
;  0574  0     !         |                             |
;  0575  0     !         |  Pointer to resource        |      NODE$V_RESOURCE
;  0576  0     !         |                             |
;  0577  0     !         +-----------------------------+ +15
;  0578  0     !         |                             |
;  0579  0     !         |       Pointer to            |      NODE$V_LIST
;  0580  0     !         |        list head            |
;  0581  0     !         +-----------------------------+ +18
;  0582  0     !         |                             |
;  0583  0     !         |          CDD                |      NODE$W_VERSION
;  0584  0     !         |       version               |
;  0585  0     !         +---------------+-------------+ +20
;  0586  0     !
;  0587  0     !       There are two types of Node Blocks Directory and Terminal.
;  0588  0     !
;  0589  0     !       Directory Node Block is made up of a Block Header + Node Block
;  0590  0     !
;  0591  0     !       Terminal Node Block is made up of a Block Header + Node Block
;  0592  0     !
;  0593  0     !-
;  0594  0
;  0595  0     LITERAL
;  0596  0         NODE$S_BLOCK_LENGTH = 20+ BLK$K_BASE;
;  0597  0
;  0598  0     MACRO
;M 0599  0         $NODE = BLOCK[NODE$S_BLOCK_LENGTH,BYTE] FIELD (BLK$Z_FIELDS,
;M 0600  0                                                        NODE$Z_FIELDS)
;  0601  0         %;
;  0602  0
;  0603  0     FIELD NODE$Z_FIELDS =
;  0604  0         SET
;  0605  0             NODE$V_ORDERED          = [1, 0, 1, 0],
;  0606  0             NODE$V_NAME             = [0+BLK$K_BASE, 0, 24, 0],
;  0607  0             NODE$V_HISTORY          = [3+BLK$K_BASE, 0, 24, 0],
;  0608  0             NODE$V_PROTECTION       = [6+BLK$K_BASE, 0, 24, 0],
```

```
;    0609  0              NODE$V_PARENT           = [9+BLK$K_BASE, 0, 24, 0],
;    0610  0              NODE$V_RESOURCE         = [12+BLK$R_BASE, 0, 24, 0],
;    0611  0              NODE$V_LIST             = [15+BLK$K_BASE, 0, 24, 0],
;    0612  0              NODE$W_VERSION          = [18+BLK$K_BASE, 0, 16, 0]
;    0613  0          TES;
;    0614  0
;    0615  0      LITERAL
;    0616  0          NODE$K_BASE         = NODE$S_BLOCK_LENGTH;
;    0617  0
;    0618  0      !+
;    0619  0      !       The following flag occurs in the BLK$B_FLAGS field of
;    0620  0      !       a directory node.
;    0621  0      !-
;    0622  0
;    0623  0      LITERAL
;    0624  0          NODE$M_ORDERED      = 1^1 - 1^0;    ! List elements are sorted by name
```

```
;  0625  0     !+
;  0626  0     !       Common Attribute Block
;  0627  0     !
;  0628  0     !
;  0629  0     !        31     24       15              0
;  0630  0     !       +----------------+---------------+ +0
;  0631  0     !       |                                |
;  0632  0     !       |         Attribute name         |      ATT$L_NAME
;  0633  0     !       |                                |
;  0634  0     !       +----------------+---------------+ +4
;  0635  0     !       |                                |
;  0636  0     !       |    Pointer to next sibling     |      ATT$V_SIBLING
;  0637  0     !       |                                |
;  0638  0     !       +--------------------------------+ +7
;  0639  0     !
;  0640  0     !       Each attribute has the common attribute block immediately following
;  0641  0     !       the universal block header.
;  0642  0     !
;  0643  0     !       There are six types of Attribute Blocks Entity Attribute Block,
;  0644  0     !       List Attribute Block, Numeric Attribute Block, Null Attribute Block,
;  0645  0     !       Short String Attribute Block, and String Attribute Block.
;  0646  0     !
;  0647  0     !       Entity Attribute Block is made up of a Block Header + Attribute Block
;  0648  0     !       + Entity Attribute Block.
;  0649  0     !
;  0650  0     !       List Attribute Block is made up of a Block Header + Attribute Block
;  0651  0     !       + List Attribute Block.
;  0652  0     !
;  0653  0     !       Numeric Attribute Block is made up of a Block Header + Attribute Block
;  0654  0     !       + Numeric Attribute BLock.
;  0655  0     !
;  0656  0     !       Null Attribute Block is made up of a Block Header + Attribute Block.
;  0657  0     !
;  0658  0     !       Short String Attribute Block is made up of a Block Header + Attribute
;  0659  0     !       Block + Short String Attribute.
;  0660  0     !
;  0661  0     !       String Attribute Block is made up of a Block Header + Attribute Block
;  0662  0     !       + String Attribute Block.
;  0663  0     !
;  0664  0     !-
;  0665  0
;  0666  0     LITERAL
;  0667  0         ATT$S_BLOCK_LENGTH = 7 + BLK$K_BASE;
;  0668  0
;  0669  0     MACRO
;M 0670  0         $ATT = BLOCK[ATT$S_BLOCK_LENGTH,BYTE] FIELD (ATT$Z_FIELDS,
;M 0671  0                                                      BLK$Z_FIELDS)
;  0672  0         %;
;  0673  0
;  0674  0     FIELD ATT$Z_FIELDS =
;  0675  0         SET
;  0676  0             ATT$L_NAME              = [0+BLK$K_BASE, 0, 32, 0],
;  0677  0             ATT$V_SIBLING           = [4+BLK$K_BASE, 0, 24, 0]
;  0678  0         TES;
;  0679  0
;  0680  0     LITERAL
;  0681  0         ATT$K_BASE          = ATT$S_BLOCK_LENGTH;
```

```
;    0682  0     !+
;    0683  0     !       Entity Attribute Block
;    0684  0     !
;    0685  0     !
;    0686  0     !        31     24       15              0
;    0687  0     !                +-----------------------+ +0
;    0688  0     !                |                       |
;    0689  0     !                |   Pointer to first    |      ENT$V_FIRST_ATT
;    0690  0     !                |      attribute        |
;    0691  0     !                +-----------------------+ +3
;    0692  0     !
;    0693  0     !       Entity Attribute Block is made up of a Block Header + Attribute Block
;    0694  0     !       + Entity Attribute Block.
;    0695  0     !
;    0696  0     !-
;    0697  0
;    0698  0     LITERAL
;    0699  0         ENT$S_BLOCK_LENGTH  = 3 + ATT$K_BASE;
;    0700  0
;    0701  0     MACRO
;  M 0702  0         $ENT = BLOCK[ENT$S_BLOCK_LENGTH,BYTE] FIELD (ATT$Z_FIELDS,
;  M 0703  0                                                      BLK$Z_FIELDS,
;  M 0704  0                                                      ENT$Z_FIELDS)
;    0705  0         %;
;    0706  0
;    0707  0     FIELD ENT$Z_FIELDS =
;    0708  0         SET
;    0709  0             ENT$V_FIRST_ATT         = [0+ATT$K_BASE, 0, 24, 0]
;    0710  0         TES;
;    0711  0
;    0712  0     LITERAL
;    0713  0         ENT$K_BASE          = ENT$S_BLOCK_LENGTH;
```

```
;    0714  0      !+
;    0715  0      !       List Attribute Block
;    0716  0      !
;    0717  0      !
;    0718  0      !        31     24       15              0
;    0719  0      !         +-----------------------------+ +0
;    0720  0      !         |                             |
;    0721  0      !         |   Pointer to first          |      LIST$V_FIRST_LST
;    0722  0      !         |     lst segment             |
;    0723  0      !         +-------+---------------------+ +3
;    0724  0      !         |       |                     |
;    0725  0      !         |       |  Total number       |      LIST$W_CELL_COUNT
;    0726  0      !         |       |    of cells         |
;    0727  0      !         |       +---------------------+ +5
;    0728  0      !         |       |                     |
;    0729  0      !       List Attribute Block is made up of a Block Header + Attribute Block
;    0730  0      !       + List Attribute Block.
;    0731  0      !
;    0732  0      !-
;    0733  0
;    0734  0      LITERAL
;    0735  0          LIST$S_BLOCK_LENGTH = 5 + ATT$K_BASE;
;    0736  0
;    0737  0      MACRO
;M   0738  0          $LIST = BLOCK[LIST$S_BLOCK_LENGTH,BYTE] FIELD (ATT$Z_FIELDS,
;M   0739  0                                                         BLK$Z_FIELDS,
;M   0740  0                                                         LIST$Z_FIELDS)
;    0741  0          %;
;    0742  0
;    0743  0      FIELD LIST$Z_FIELDS =
;    0744  0          SET
;    0745  0              LIST$V_FIRST_LST        = [0+ATT$K_BASE, 0, 24, 0],
;    0746  0              LIST$W_CELL_COUNT       = [3+ATT$K_BASE, 0, 16, 0]
;    0747  0          TES;
;    0748  0
;    0749  0      LITERAL
;    0750  0          LIST$K_BASE          = LIST$S_BLOCK_LENGTH;
```

```
;    0751  0     !+
;    0752  0     !       Numeric Attribute Block
;    0753  0     !
;    0754  0     !
;    0755  0     !       31       24       15                0
;    0756  0     !       +---------------------------------+ +0
;    0757  0     !       |                                 |
;    0758  0     !       |     Numeric Attribute Value     |      NAT$L_VALUE
;    0759  0     !       |                                 |
;    0760  0     !       +---------------------------------+ +4
;    0761  0     !
;    0762  0     !
;    0763  0     !       Numeric Attribute Block is made up of a Block Header + Attribute Block
;    0764  0     !       + Numeric Attribute BLock.
;    0765  0     !
;    0766  0     !       Numeric attribute block contains the
;    0767  0     !       value of the numeric attribute.
;    0768  0     !
;    0769  0     !-
;    0770  0
;    0771  0     LITERAL
;    0772  0         NAT$S_BLOCK_LENGTH  = 4 + ATT$K_BASE;
;    0773  0
;    0774  0     MACRO
;M   0775  0         $NAT = BLOCK[NAT$S_BLOCK_LENGTH,BYTE] FIELD (ATT$Z_FIELDS,
;M   0776  0                                                      BLK$Z_FIELDS,
;M   0777  0                                                      NAT$Z_FIELDS)
;    0778  0         %;
;    0779  0
;    0780  0     FIELD NAT$Z_FIELDS =
;    0781  0         SET
;    0782  0             NAT$L_VALUE             = [0+ATT$K_BASE,0,32,0]
;    0783  0         TES;
;    0784  0
;    0785  0     LITERAL
;    0786  0         NAT$K_BASE                  = NAT$S_BLOCK_LENGTH;
```

```
;    0787  0     !+
;    0788  0     !       Short String Attribute Block
;    0789  0     !
;    0790  0     !       31                                  0
;    0791  0     !                                +---------+ +0   SSA$T_STRING
;    0792  0     !                                |         |
;    0793  0     !                                | String  |
;    0794  0     !                                /         \
;    0795  0     !                                \         /
;    0796  0     !                                |         |
;    0797  0     !                                |         |
;    0798  0     !                                +---------+ +n
;    0799  0     !
;    0800  0     !       Short String Attribute Block is made up of a Block Header + Attribute
;    0801  0     !       Block + Short String Attribute.
;    0802  0     !
;    0803  0     !       Strings whose total length is between 0 and 255 bytes are usually
;    0804  0     !       stored using the short string attribute block.  If a string is
;    0805  0     !       too long to be stored in an SSA, then it is stored using a normal
;    0806  0     !       string attribute block (STR).
;    0807  0     !
;    0808  0     !-
;    0809  0
;    0810  0     LITERAL
;    0811  0         SSA$S_BLOCK_LENGTH  = 0 + ATT$K_BASE;
;    0812  0
;    0813  0     MACRO
;M   0814  0         $SSA = BLOCK[SSA$S_BLOCK_LENGTH,BYTE] FIELD (ATT$Z_FIELDS,
;M   0815  0                                                      BLK$Z_FIELDS,
;M   0816  0                                                      SSA$Z_FIELDS)
;    0817  0         %;
;    0818  0
;    0819  0     FIELD SSA$Z_FIELDS =
;    0820  0         SET
;    0821  0             SSA$T_STRING            = [0+ATT$K_BASE, 0, 0, 0]
;    0822  0         TES;
;    0823  0
;    0824  0     LITERAL
;    0825  0         SSA$K_BASE          = SSA$S_BLOCK_LENGTH;
;    0826  0
```

```
;    0827  0     !+
;    0828  0     !       String Attribute Block
;    0829  0     !
;    0830  0     !
;    0831  0     !        31     24       15              0
;    0832  0     !          +-----------------------+ +0
;    0833  0     !          |                       |
;    0834  0     !          |   Pointer to first    |      STR$V_STRING
;    0835  0     !          |    string segment     |
;    0836  0     !          +-------+---------------+ +3
;    0837  0     !                  |               |
;    0838  0     !                  | Total string  |      STR$W_LENGTH
;    0839  0     !                  |    length     |
;    0840  0     !                  +---------------+ +5
;    0841  0     !
;    0842  0     !       String Attribute Block is made up of a Block Header + Attribute Block
;    0843  0     !       + String Attribute Block.
;    0844  0     !
;    0845  0     !       The string is broken into one or more segments.  The string
;    0846  0     !       segments are stored in SEG blocks.
;    0847  0     !-
;    0848  0
;    0849  0     LITERAL
;    0850  0         STR$S_BLOCK_LENGTH  = 5 + ATT$K_BASE;
;    0851  0
;    0852  0     MACRO
;M   0853  0         $STR = BLOCK[STR$S_BLOCK_LENGTH,BYTE] FIELD (ATT$Z_FIELDS,
;M   0854  0                                                      BLK$Z_FIELDS,
;M   0855  0                                                      STR$Z_FIELDS)
;    0856  0         %;
;    0857  0
;    0858  0     FIELD STR$Z_FIELDS =
;    0859  0         SET
;    0860  0             STR$V_STRING            = [0+ATT$K_BASE, 0, 24, 0],
;    0861  0             STR$W_LENGTH            = [3+ATT$K_BASE, 0, 16, 0]
;    0862  0         TES;
;    0863  0
;    0864  0     LITERAL
;    0865  0         STR$K_BASE          = STR$S_BLOCK_LENGTH;
```

```
;    0866  0     !+
;    0867  0     !       LST Segment Block
;    0868  0     !
;    0869  0     !
;    0870  0     !        31     24       15              0
;    0871  0     !                +-----------------------+ +0
;    0872  0     !                |                       |
;    0873  0     !                |  Pointer to next lst  |      LST$V_NEXT_LST
;    0874  0     !                |                       |
;    0875  0     !                +-------+---------------+ +3
;    0876  0     !                        |               |
;    0877  0     !                        |Number of cells|      LST$W_CELL_COUNT
;    0878  0     !                        |      here     |
;    0879  0     !                        +---------------+ +5   LST$T_CELLS
;    0880  0     !
;    0881  0     !       There are two types of LST Segment Blocks Entity List Block and
;    0882  0     !       String List Attribute Block.
;    0883  0     !
;    0884  0     !       Entity List Block is made up of a Block Header + LST Block +
;    0885  0     !       Entity List Block.
;    0886  0     !
;    0887  0     !       String List Attribute Block is made up of a Block Header + LST
;    0888  0     !       Block + String List Attribute Block.
;    0889  0     !
;    0890  0     !-
;    0891  0
;    0892  0     LITERAL
;    0893  0         LST$S_BLOCK_LENGTH  = 5 + BLK$K_BASE;
;    0894  0
;    0895  0     MACRO
; M  0896  0         $LST = BLOCK[LST$S_BLOCK_LENGTH,BYTE] FIELD (BLK$Z_FIELDS,
; M  0897  0                                                      LST$Z_FIELDS)
;    0898  0         %;
;    0899  0
;    0900  0
;    0901  0     FIELD LST$Z_FIELDS =
;    0902  0         SET
;    0903  0             LST$V_NEXT_LST          = [0 + BLK$K_BASE, 0, 24, 0],
;    0904  0             LST$W_CELL_COUNT        = [3 + BLK$K_BASE, 0, 16, 0],
;    0905  0             LST$T_CELLS             = [5 + BLK$K_BASE, 0, 0, 0]
;    0906  0         TES;
;    0907  0
;    0908  0     LITERAL
;    0909  0         LST$K_BASE          = LST$S_BLOCK_LENGTH;
```

```
;   0910  0     !+
;   0911  0     !       Entity List Block
;   0912  0     !
;   0913  0     !
;   0914  0     !        31     24        15                0
;   0915  0     !                +------------------------+ +0
;   0916  0     !                |                        |
;   0917  0     !                |  Pointer to first      |      ELST$V_ATT
;   0918  0     !                |      attribute         |
;   0919  0     !                +------------------------+ +3
;   0920  0     !
;   0921  0     !       Entity List Block is made up of a Block Header + LST Block +
;   0922  0     !       Entity List Block.
;   0923  0     !
;   0924  0     !-
;   0925  0
;   0926  0     LITERAL
;   0927  0         ELST$S_BLOCK_LENGTH = 3;
;   0928  0
;   0929  0     MACRO
; M 0930  0         $ELST = BLOCK[ELST$S_BLOCK_LENGTH,BYTE] FIELD (ELST$Z_FIELDS)
;   0931  0         %;
;   0932  0
;   0933  0
;   0934  0     FIELD ELST$Z_FIELDS =
;   0935  0         SET
;   0936  0             ELST$V_ATT      = [0, 0, 24, 0]
;   0937  0         TES;
;   0938  0
;   0939  0     LITERAL
;   0940  0         ELST$K_BASE         = ELST$S_BLOCK_LENGTH;
;   0941  0
;   0942  0
;   0943  0     STRUCTURE
;   0944  0         ELST$ELM[I] =
;   0945  0             (ELST$ELM + LST$K_BASE + (I * ELST$S_BLOCK_LENGTH));
```

```
;  0946  0     !+
;  0947  0     !       String List Attribute Block
;  0948  0     !
;  0949  0     !
;  0950  0     !        31     24       15              0
;  0951  0     !                +-----------------------+ +0
;  0952  0     !                |   Pointer to first    |      SLST$V_STRING
;  0953  0     !                |    string segment     |
;  0954  0     !                |    or text block      |
;  0955  0     !                +-------+---------------+ +3
;  0956  0     !                        |               |
;  0957  0     !                        | Total string  |      SLST$W_LENGTH
;  0958  0     !                        |    length     |
;  0959  0     !                        +---------------+ +5   SLST$T_STRING
;  0960  0     !
;  0961  0     !       String List Attribute Block is made up of a Block Header + LST
;  0962  0     !       Block + String List Attribute Block.
;  0963  0     !
;  0964  0     !-
;  0965  0
;  0966  0     LITERAL
;  0967  0         SLST$S_BLOCK_LENGTH = 5;
;  0968  0
;  0969  0     MACRO
;M 0970  0         $SLST = BLOCK[SLST$S_BLOCK_LENGTH,BYTE] FIELD (SLST$Z_FIELDS)
;  0971  0         %;
;  0972  0
;  0973  0
;  0974  0     FIELD SLST$Z_FIELDS =
;  0975  0         SET
;  0976  0             SLST$V_STRING           = [0, 0, 24, 0],
;  0977  0             SLST$W_LENGTH           = [3, 0, 16, 0],
;  0978  0             SLST$T_STRING           = [5, 0, 0, 0]
;  0979  0         TES;
;  0980  0
;  0981  0     LITERAL
;  0982  0         SLST$K_BASE         = SLST$S_BLOCK_LENGTH;
;  0983  0
;  0984  0
;  0985  0     STRUCTURE
;  0986  0         SLST$ELM[I] =
;  0987  0             (SLST$ELM + LST$K_BASE + (I * SLST$S_BLOCK_LENGTH));
```

```
;  0988  0     !+
;  0989  0     !       String Segment Block
;  0990  0     !
;  0991  0     !        31     24       15              0
;  0992  0     !                +-----------------------+ +0
;  0993  0     !                |                       |
;  0994  0     !                |    Pointer to next    |      SEG$V_NEXT
;  0995  0     !                |    string segment     |
;  0996  0     !                +-----------------------+ +3   SEG$T_STRING
;  0997  0     !
;  0998  0     !       String Segment Block is made up of a Block Header + String
;  0999  0     !       Segment Block.
;  1000  0     !
;  1001  0     !       Each string attribute points to zero or more string segment
;  1002  0     !       blocks.  Each block contains a portion of the whole string.
;  1003  0     !       The final string is found by catenating all the string
;  1004  0     !       segments together.
;  1005  0     !
;  1006  0     !-
;  1007  0
;  1008  0     LITERAL
;  1009  0         SEG$S_BLOCK_LENGTH  = 3 + BLK$K_BASE;
;  1010  0
;  1011  0     MACRO
;M 1012  0         $SEG = BLOCK[SEG$S_BLOCK_LENGTH,BYTE] FIELD (BLK$Z_FIELDS,
;M 1013  0                                                      SEG$Z_FIELDS)
;  1014  0         %;
;  1015  0
;  1016  0     FIELD SEG$Z_FIELDS =
;  1017  0         SET
;  1018  0             SEG$V_NEXT              = [0+BLK$K_BASE, 0, 24, 0],
;  1019  0             SEG$T_STRING            = [3+BLK$K_BASE, 0, 0, 0]
;  1020  0         TES;
;  1021  0
;  1022  0     LITERAL
;  1023  0         SEG$K_BASE          = SEG$S_BLOCK_LENGTH;
```

```
;    1024  0     !+
;    1025  0     !       Text Block
;    1026  0     !
;    1027  0     !       31                              0
;    1028  0     !                                +--------+ +0   TEXT$T_STRING
;    1029  0     !                                |        |
;    1030  0     !                                |  Text  |
;    1031  0     !                                /        \
;    1032  0     !                                \        /
;    1033  0     !                                |        |
;    1034  0     !                                |        |
;    1035  0     !                                +--------+ +n
;    1036  0     !
;    1037  0     !       Text Block is made up of a Block Header + Text BLock.
;    1038  0     !-
;    1039  0
;    1040  0     LITERAL
;    1041  0         TEXT$S_BLOCK_LENGTH = 0 + BLK$K_BASE;
;    1042  0
;    1043  0     MACRO
;M   1044  0         $TEXT = BLOCK[TEXT$S_BLOCK_LENGTH,BYTE] FIELD (BLK$Z_FIELDS,
;M   1045  0                                                        TEXT$Z_FIELDS)
;    1046  0         %;
;    1047  0
;    1048  0     FIELD TEXT$Z_FIELDS =
;    1049  0         SET
;    1050  0             TEXT$T_STRING           = [0+BLK$K_BASE, 0, 0, 0]
;    1051  0         TES;
;    1052  0
;    1053  0     LITERAL
;    1054  0         TEXT$K_BASE         = TEXT$S_BLOCK_LENGTH;
;    1055  0
```

```
;  1056  0     !+
;  1057  0     !       Access Control List Entry (ACL)
;  1058  0     !
;  1059  0     !        31              15              0
;  1060  0     !       +---------------+---------------+ +0
;  1061  0     !       |                               |
;  1062  0     !       |        Ptr to next ACL        |      ACL$V_NEXT
;  1063  0     !       |                               |
;  1064  0     !       +---------------+---------------+ +3
;  1065  0     !       |                               |
;  1066  0     !       |     Rights to be granted      |      ACL$L_GRANT
;  1067  0     !       |                               |
;  1068  0     !       +-------------------------------+ +7
;  1069  0     !       |                               |
;  1070  0     !       |      Rights to be denied      |      ACL$L_DENY
;  1071  0     !       |                               |
;  1072  0     !       +-------------------------------+ +11
;  1073  0     !       |                               |
;  1074  0     !       |     Rights to be banished     |      ACL$L_BANISH
;  1075  0     !       |                               |
;  1076  0     !       +---------------+---------------+ +15
;  1077  0     !       |               |               |
;  1078  0     !       |  UIC group #  |  UIC member #  |      ACL$W_UIC_MEMBER
;  1079  0     !       |               |               |      ACL$W_UIC_GROUP
;  1080  0     !       +---------------+---------------+ +19
;  1081  0     !       |                               |
;  1082  0     !       |      Ptr to first ACLC block  |      ACL$V_FIRST_ACLC
;  1083  0     !       |                               |
;  1084  0     !       +-------------------------------+ +22
;  1085  0     !
;  1086  0     !       Each node has an Access Control List made up of zero or more
;  1087  0     !       Access Control List Entries (ACL).
;  1088  0     !
;  1089  0     !       The ACL block is appended to a BLK to make an access control
;  1090  0     !       list entry.
;  1091  0     !-
;  1092  0
;  1093  0     LITERAL
;  1094  0         ACL$S_BLOCK_LENGTH = 22+BLK$K_BASE;
;  1095  0
;  1096  0     MACRO
;M 1097  0         $ACL = BLOCK[ACL$S_BLOCK_LENGTH, BYTE] FIELD (BLK$Z_FIELDS,
;M 1098  0                                                       ACL$Z_FIELDS)
;  1099  0         %;
;  1100  0
;  1101  0     FIELD   ACL$Z_FIELDS =
;  1102  0         SET
;  1103  0             ACL$V_NEXT              = [0+BLK$K_BASE, 0, 24, 0],
;  1104  0             ACL$L_GRANT             = [3+BLK$K_BASE, 0, 32, 0],
;  1105  0             ACL$L_DENY              = [7+BLK$K_BASE, 0, 32, 0],
;  1106  0             ACL$L_BANISH            = [11+BLK$K_BASE, 0, 32, 0],
;  1107  0             ACL$W_UIC_MEMBER        = [15+BLK$K_BASE, 0, 16, 0],
;  1108  0             ACL$W_UIC_GROUP         = [17+BLK$K_BASE, 0, 16, 0],
;  1109  0             ACL$V_FIRST_ACLC        = [19+BLK$K_BASE, 0, 24, 0]
;  1110  0         TES;
;  1111  0
;  1112  0     LITERAL
```

```
;   1113  0          ACL$K_BASE                  = ACL$S_BLOCK_LENGTH;
```

```
;  1114  0     !+
;  1115  0     !       Access Control List Criterion (ACLC)
;  1116  0     !
;  1117  0     !        31      .        15              0
;  1118  0     !       +-------+--------+---------------+ +0
;  1119  0     !       |       |        |               |
;  1120  0     !       | Type  |   Ptr to next ACLC     |      ACLC$V_NEXT
;  1121  0     !       |       |        |               |      ACLC$B_TYPE
;  1122  0     !       +-------+--------+---------------+ +4   ACLC$T_STRING
;  1123  0     !
;  1124  0     !       Each ACL may have one or more Access Control List Criterion
;  1125  0     !       (ACLC) chained from it.
;  1126  0     !
;  1127  0     !       Each ACLC specifies one user identification criterion for
;  1128  0     !       the ACL entry.  The user must match all the criteria for
;  1129  0     !       the ACL entry to apply to him.
;  1130  0     !
;  1131  0     !       An ACLC is appended to a BLK to form the criterion block.
;  1132  0     !-
;  1133  0
;  1134  0     LITERAL
;  1135  0         ACLC$S_BLOCK_LENGTH = 4+BLK$K_BASE;
;  1136  0
;  1137  0     MACRO
;M 1138  0         $ACLC = BLOCK[ACLC$S_BLOCK_LENGTH, BYTE] FIELD (BLK$Z_FIELDS,
;M 1139  0                                                         ACLC$Z_FIELDS)
;  1140  0         %;
;  1141  0
;  1142  0     FIELD   ACLC$Z_FIELDS =
;  1143  0         SET
;  1144  0             ACLC$V_NEXT             = [0+BLK$K_BASE, 0, 24, 0],
;  1145  0             ACLC$B_TYPE             = [3+BLK$K_BASE, 0, 8, 0],
;  1146  0             ACLC$T_STRING           = [4+BLK$K_BASE, 0, 0, 0]
;  1147  0         TES;
;  1148  0
;  1149  0     LITERAL
;  1150  0         ACLC$K_BASE                 = ACLC$S_BLOCK_LENGTH;
```

```
;    1151  0      !+
;    1152  0      !       String Descriptor
;    1153  0      !
;    1154  0      !        31               15               0
;    1155  0      !       +--------+--------+----------------+ +0
;    1156  0      !       |        |        |                |
;    1157  0      !       | Class  | Data   |    Length      |
;    1158  0      !       |        | type   |                |
;    1159  0      !       +--------+--------+----------------+
;    1160  0      !       |                                  |
;    1161  0      !       |       Pointer to data            |
;    1162  0      !       |                                  |
;    1163  0      !       +----------------------------------+
;    1164  0      !
;    1165  0      !-
;    1166  0
;    1167  0      MACRO
;  M 1168  0          $DSC = BLOCK[8,BYTE]
;    1169  0          %;
```

```
;  1170  0      !+
;  1171  0      !       User Control Block (UCB)
;  1172  0      !
;  1173  0      !
;  1174  0      !        31               15               0
;  1175  0      !       +--------------------------------+ +0
;  1176  0      !       |                                |
;  1177  0      !       |       Pointer to next UCB      |      UCB$A_NEXT
;  1178  0      !       |                                |
;  1179  0      !       +----------------+---------------+ +4
;  1180  0      !       |                |               |
;  1181  0      !       |  Last User     |  First User   |      UCB$W_FIRST
;  1182  0      !       |  Number Mapped |  Number Mapped|      UCB$W_LAST
;  1183  0      !       +----------------+---------------+ +8   UCB$V_CCB_BASE
;  1184  0      !       |                                |
;  1185  0      !       |                                |
;  1186  0      !       |                                |
;  1187  0      !       +/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\+      UCB$A_CCB
;  1188  0      !       |                                |
;  1189  0      !       |                                |
;  1190  0      !       |                                |
;  1191  0      !       +--------------------------------+ +24
;  1192  0      !
;  1193  0      !       A user's context pointer is an origin 1 index into the CCBs in
;  1194  0      !       the UCB list.  Each UCB can point to UCB$K_CCB number of CCBs.
;  1195  0      !       If the user passes a context number that doesn't map to an
;  1196  0      !       active CCB, we tell him it's an invalid context pointer.
;  1197  0      !
;  1198  0      !       Except in some extreme case, any image will probably never have
;  1199  0      !       more than 4 users active at any one time. But we can handle more!
;  1200  0      !-
;  1201  0
;  1202  0      LITERAL
;  1203  0          UCB$K_CCB_BASE              = 8,
;  1204  0          UCB$K_CCB                   = 4,    ! Number of CCBs/UCB
;  1205  0          UCB$S_BLOCK_LENGTH          = UCB$K_CCB_BASE + UCB$K_CCB * 4;
;  1206  0
;  1207  0      MACRO
; M 1208  0          $UCB = BLOCK[UCB$S_BLOCK_LENGTH,BYTE] FIELD (UCB$Z_FIELDS)
;  1209  0          %;
;  1210  0
;  1211  0      FIELD   UCB$Z_FIELDS =
;  1212  0          SET
;  1213  0              UCB$A_NEXT              = [0, 0, 32, 0],
;  1214  0              UCB$W_FIRST             = [4, 0, 16, 0],
;  1215  0              UCB$W_LAST              = [6, 0, 16, 0],
;  1216  0              UCB$V_CCB               = [8, 0, 0, 0],
;  1217  0              UCB$A_CCB               = [0, 0, 32, 0]
;  1218  0          TES;
;  1219  0
;  1220  0      LITERAL
;  1221  0          UCB$C_FIRST = BLOCK[0, UCB$W_FIRST; UCB$S_BLOCK_LENGTH, BYTE];
;  1222  0
;  1223  0      STRUCTURE
;  1224  1          UCB$CCB[I, O, P, S, E] = (UCB$CCB+UCB$K_CCB_BASE+
;  1225  0                                    (I-.(UCB$CCB+UCB$C_FIRST)<0,16,0>)*4+0)<P,S,E>;
```

```
;    1226  0      !+
;    1227  0      !       Pool Header Block (PHB)
;    1228  0      !
;    1229  0      !        31              15              0
;    1230  0      !       +-------+-------+-------+-------+ +0
;    1231  0      !       |       | Number| Low   |       |     PHB$B_POOL_TYPE
;    1232  0      !       |       |  of   | slot  | Pool  |     PHB$B_LOW_SLOT
;    1233  0      !       |       | slots | number| type  |     PHB$B_SLOTS
;    1234  0      !       +-------+-------+-------+-------+ +4
;    1235  0      !       |                               |
;    1236  0      !       |   Ptr to last MCB in the pool |     PHB$A_LAST_MCB
;    1237  0      !       |                               |
;    1238  0      !       +-------------------------------+ +8
;    1239  0      !       |                               |
;    1240  0      !       |    Number of bytes in each    |     PHB$L_EXTENT_SIZE
;    1241  0      !       |       secondary extent        |
;    1242  0      !       +-------------------------------+ +12  PHB$V_FREE_BLOCK_LIST
;    1243  0      !
;    1244  0      !       Pools are used to provide space for various in-core block types.
;    1245  0      !       Blocks that are related are usually allocated from the same pool.
;    1246  0      !       This is done because pools provide good locality of reference and
;    1247  0      !       this allocation scheme reduces page faults.
;    1248  0      !
;    1249  0      !       Each pool consists of one or more extents.  Each extent consists of
;    1250  0      !       a Memory Control Block and the rest of the space allocated to the
;    1251  0      !       extent.  Each pools has a Pool Header Block associated with it.
;    1252  0      !       This block identifies the MCB list, as well as contains the pool's
;    1253  0      !       free block list.
;    1254  0      !
;    1255  0      !           The free block list is a vector of linked lists.  Each slot in the
;    1256  0      !           vector corresponds to a block type.  When a block is freed, it is
;    1257  0      !           linked into its associated free list.  When a block is requested,
;    1258  0      !           its free list is checked to see if it is non-empty.  If so, then
;    1259  0      !           a block is allocated from it.  Otherwise, a block is allocated from
;    1260  0      !           one of the pool's extents.
;    1261  0      !-
;    1262  0
;    1263  0      LITERAL
;    1264  0          PHB$S_BLOCK_LENGTH          = 12;
;    1265  0
;    1266  0      MACRO
;M   1267  0          $PHB = BLOCK[PHB$S_BLOCK_LENGTH, BYTE]  FIELD (PHB$Z_FIELDS)
;    1268  0          %;
;    1269  0
;    1270  0      FIELD   PHB$Z_FIELDS =
;    1271  0          SET
;    1272  0              PHB$B_POOL_TYPE         = [0, 0, 8, 0],
;    1273  0              PHB$B_LOW_SLOT          = [1, 0, 8, 0],
;    1274  0              PHB$B_SLOTS             = [2, 0, 8, 0],
;    1275  0              PHB$A_LAST_MCB          = [4, 0, 32, 0],
;    1276  0              PHB$L_EXTENT_SIZE       = [8, 0, 32, 0],
;    1277  0              PHB$V_FREE_BLOCK_LIST   = [12, 0, 0, 0]
;    1278  0          TES;
```

```
;    1279  0      !+
;    1280  0      !       Memory Control Block (MCB)
;    1281  0      !
;    1282  0      !        31               15               0
;    1283  0      !       +---------------------------------+ +0
;    1284  0      !       |                                 |
;    1285  0      !       |  Pointer to prior memory extent |      MCB$A_PRIOR
;    1286  0      !       |                                 |
;    1287  0      !       +-----------------+---------------+ +4
;    1288  0      !       |                 |               |
;    1289  0      !       |  Remaining size |  Size of this |      MCB$W_ALLOC_SIZE
;    1290  0      !       |  of this extent |    extent     |      MCB$W_FREE_SIZE
;    1291  0      !       +-----------------+---------------+ +8
;    1292  0      !
;    1293  0      !       Each pool has one or more Memory Control Blocks associated with it.
;    1294  0      !
;    1295  0      !       This list serves two purposes:
;    1296  0      !
;    1297  0      !       1)  It points to each memory extent we asked LIB$GET_VM for.
;    1298  0      !
;    1299  0      !       2)  It keeps track of memory that has been allocated for
;    1300  0      !           the pool, but never used.
;    1301  0      !-
;    1302  0
;    1303  0      LITERAL
;    1304  0          MCB$S_BLOCK_LENGTH = 8;
;    1305  0
;    1306  0      MACRO
;M   1307  0          $MCB = BLOCK[MCB$S_BLOCK_LENGTH,BYTE] FIELD (MCB$Z_FIELDS)
;    1308  0          %;
;    1309  0
;    1310  0      FIELD   MCB$Z_FIELDS =
;    1311  0          SET
;    1312  0              MCB$A_PRIOR             = [0, 0, 32, 0],
;    1313  0              MCB$W_ALLOC_SIZE        = [4, 0, 16, 0],
;    1314  0              MCB$W_FREE_SIZE         = [6, 0, 16, 0]
;    1315  0          TES;
```

```
;  1316  0     !+
;  1317  0     !       Hash Control Entry (HCE)
;  1318  0     !
;  1319  0     !        31               15               0
;  1320  0     !       +---------------------------------+ +0
;  1321  0     !       |                                 |
;  1322  0     !       |     Forward homonym pointer     |      HCE$A_NEXT
;  1323  0     !       |                                 |
;  1324  0     !       +---------------------------------+ +4
;  1325  0     !       |                                 |
;  1326  0     !       |    Backwards homonym pointer    |      HCE$A_PRIOR
;  1327  0     !       |                                 |
;  1328  0     !       +----------------+----------------+ +8
;  1329  0     !       |                 |                |
;  1330  0     !       | # blocks ever   | # times bucket |      HCE$W_HITS
;  1331  0     !       |stored in bucket |   accessed     |      HCE$W_BLOCKS
;  1332  0     !       +----------------+----------------+ +12
;  1333  0     !       |                 |                |
;  1334  0     !       | Max blocks ever | # blocks now   |      HCE$W_CUR_LENGTH
;  1335  0     !       |  in bucket chain|in bucket chain |      HCE$W_MAX_LENGTH
;  1336  0     !       +----------------+----------------+ +16
;  1337  0     !
;  1338  0     !       HCEs are found in each user's CCB.  Each HCE forms one hash
;  1339  0     !       bucket.  We keep some statistics on the buckets in an attempt
;  1340  0     !       to tune the hash function.
;  1341  0     !-
;  1342  0
;  1343  0     LITERAL
;  1344  0         HCE$S_BLOCK_LENGTH = 16;
;  1345  0
;  1346  0     MACRO
;M 1347  0         $HCE = BLOCK[HCE$S_BLOCK_LENGTH,BYTE] FIELD (HCE$Z_FIELDS)
;  1348  0         %;
;  1349  0
;  1350  0     FIELD  HCE$Z_FIELDS =
;  1351  0         SET
;  1352  0             HCE$A_NEXT              = [0, 0, 32, 0],
;  1353  0             HCE$A_PRIOR             = [4, 0, 32, 0],
;  1354  0             HCE$W_HITS              = [8, 0, 16, 0],
;  1355  0             HCE$W_BLOCKS            = [10, 0, 16, 0],
;  1356  0             HCE$W_CUR_LENGTH        = [12, 0, 16, 0],
;  1357  0             HCE$W_MAX_LENGTH        = [14, 0, 16, 0]
;  1358  0         TES;
;  1359  0
;  1360  0     LITERAL
;  1361  0         HCE$K_HOMONYM_LIST          = BLOCK[0, HCE$A_NEXT; ,BYTE];
```

```
;    1362  0      !+
;    1363  0      !       Hash Control Block (HCB)
;    1364  0      !
;    1365  0      !        31              15              0
;    1366  0      !       +--------------------------------+ +0
;    1367  0      !       |                                |
;    1368  0      !       |     Forward homonym pointer    |      HCB$A_NEXT
;    1369  0      !       |                                |
;    1370  0      !       +--------------------------------+ +4
;    1371  0      !       |                                |
;    1372  0      !       |    Backwards homonym pointer   |      HCB$A_PRIOR
;    1373  0      !       |                                |
;    1374  0      !       +--------------------------------+ +8
;    1375  0      !       |                                |
;    1376  0      !       |           Hash key             |      HCB$V_HASH_KEY
;    1377  0      !       |                                |
;    1378  0      !       +---------------+----------------+ +12
;    1379  0      !       |                                |
;    1380  0      !       |        Rest of hash key        |
;    1381  0      !       |                                |
;    1382  0      !       +---------------+----------------+ +16
;    1383  0      !       |                                |
;    1384  0      !       |    Pointer to object block     |      HCB$A_BLOCK
;    1385  0      !       |                                |
;    1386  0      !       +--------------------------------+ +20
;    1387  0      !
;    1388  0      !-
;    1389  0
;    1390  0      LITERAL
;    1391  0          HCB$K_PCB_NUMBER    = 1,    ! Hash type is page number
;    1392  0          HCB$K_LCCB_CODE     = 2,    ! Hash type is location code
;    1393  0          HCB$K_LCCB_ADDRESS  = 3,    ! Hash type is entity disk address
;    1394  0          HCB$K_KEY_LENGTH    = 8,    ! Size of a hash key
;    1395  0          HCB$S_BLOCK_LENGTH  = 20;
;    1396  0
;    1397  0      MACRO
;  M 1398  0          $HCB = BLOCK[HCB$S_BLOCK_LENGTH,BYTE] FIELD (HCB$Z_FIELDS)
;    1399  0          %;
;    1400  0
;    1401  0      FIELD  HCB$Z_FIELDS =
;    1402  0          SET
;    1403  0              HCB$A_NEXT              = [0, 0, 32, 0],
;    1404  0              HCB$A_PRIOR             = [4, 0, 32, 0],
;    1405  0              HCB$V_HASH_KEY          = [8, 0, 0, 0],
;    1406  0              HCB$A_BLOCK             = [16, 0, 32, 0]
;    1407  0          TES;
;    1408  0
;    1409  0      LITERAL
;    1410  0          HCB$K_HOMONYM_LIST          = BLOCK[0, HCB$A_NEXT; , BYTE];
```

```
1411  0  !+
1412  0  !       Context Control Block (CCB)
1413  0  !
1414  0  !
1415  0  !        31               15               0
1416  0  !       +----------------+----------------+ +0
1417  0  !       |                |                |
1418  0  !       |                |     Flags      |      CCB$W_FLAGS
1419  0  !       |                |                |
1420  0  !       +----------------+----------------+ +4
1421  0  !       |                                 |
1422  0  !       |     Pointer to first LCB        |      CCB$A_FIRST_LOCK
1423  0  !       |     in temporary lock list      |
1424  0  !       +---------------------------------+ +8
1425  0  !       |                                 |
1426  0  !       |     Pointer to last LCB         |      CCB$A_LAST_LOCK
1427  0  !       |     in temporary lock list      |
1428  0  !       +----------------+----------------+ +12
1429  0  !       |                |                |
1430  0  !       |    User ID     |  User ID for   |      CCB$W_LOCK_ID
1431  0  !       |                |  lock manager  |      CCB$W_USER_ID
1432  0  !       +----------------+----------------+ +16
1433  0  !       |                                 |
1434  0  !       |       Next location code        |      CCB$L_NEXT_LCC
1435  0  !       |     number to be assigned       |
1436  0  !       +---------------------------------+ +20
1437  0  !       |                                 |
1438  0  !       |     Ptr to NCB for file's       |      CCB$$A_CDD$TOP_NCB
1439  0  !       |         CDD$TOP node            |
1440  0  !       +---------------------------------+ +24
1441  0  !       |                                 |
1442  0  !       |       NCB of CDD$LOGIN          |      CCB$A_LOGIN_NCB
1443  0  !       |           node                  |
1444  0  !       +---------------------------------+ +28
1445  0  !       |                                 |
1446  0  !       |     Ptr to current FCB          |      CCB$A_CURRENT_FCB
1447  0  !       |         for user                |
1448  0  !       +---------------------------------+ +32
1449  0  !       |                                 |
1450  0  !       |     Ptr to highest cluster      |      CCB$A_FIRST_CIB
1451  0  !       |         in cache                |
1452  0  !       +---------------------------------+ +36
1453  0  !       |                                 |
1454  0  !       |     Ptr to last highest         |      CCB$A_LAST_CIB
1455  0  !       |      cluster in cache           |
1456  0  !       +---------------------------------+ +40
1457  0  !       |                                 |
1458  0  !       |     Status of current           |      CCB$L_STATUS
1459  0  !       |      transaction                |
1460  0  !       +---------------------------------+ +44
1461  0  !       |                                 |
1462  0  !       |   Ptr to CCB's pool header      |      CCB$A_POOL
1463  0  !       |                                 |
1464  0  !       +---------------------------------+ +48
1465  0  !       |                                 |
1466  0  !       |       Ptr to first              |      CCB$A_FIRST_SPB
1467  0  !       |  Security Preservation Block    |
```

```
;  1468  0     !       +---------------------------------+ +52
;  1469  0     !       |                                 |
;  1470  0     !       |          Ptr to last            |      CCB$A_LAST_SPB
;  1471  0     !       |     Security Preservation Block |
;  1472  0     !       +---------------------------------+ +56  CCB$V_HASH_BASE
;  1473  0     !       |                                 |
;  1474  0     !       |                                 |
;  1475  0     !       |                                 |
;  1476  0     !       +/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\+
;  1477  0     !       |                                 |
;  1478  0     !       |                                 |
;  1479  0     !       |                                 |
;  1480  0     !       +---------------------------------+
;  1481  0     !
;  1482  0     !       The hash table consists of a number of hash entries.
;  1483  0     !       See the HCE description for the format of these entries.
;  1484  0     !-
;  1485  0
;  1486  0     LITERAL
;  1487  0         CCB$K_HASH_TABLE_SIZE = 151,
;  1488  0         CCB$S_BLOCK_LENGTH    = 56 + (CCB$K_HASH_TABLE_SIZE * HCE$S_BLOCK_LENGTH);
;  1489  0
;  1490  0     MACRO
;M 1491  0         $CCB = BLOCK[CCB$S_BLOCK_LENGTH,BYTE] FIELD (CCB$Z_FIELDS)
;  1492  0         %;
;  1493  0
;  1494  0     FIELD  CCB$Z_FIELDS =
;  1495  0         SET
;  1496  0             CCB$W_FLAGS             = [0, 0, 16, 0],
;  1497  0             CCB$V_CORRUPT           = [0, 0, 1, 0],         ! Stream is corrupt
;  1498  0             CCB$A_FIRST_LOCK        = [4, 0, 32, 0],
;  1499  0             CCB$A_LAST_LOCK         = [8, 0, 32, 0],
;  1500  0             CCB$W_LOCK_ID           = [12, 0, 16, 0],
;  1501  0             CCB$W_USER_ID           = [14, 0, 16, 0],
;  1502  0             CCB$L_NEXT_LCC          = [16, 0, 32, 0],
;  1503  0             CCB$A_CDDSTOP_NCB       = [20, 0, 32, 0],
;  1504  0             CCB$A_LOGIN_NCB         = [24, 0, 32, 0],
;  1505  0             CCB$A_CURRENT_FCB       = [28, 0, 32, 0],
;  1506  0             CCB$A_FIRST_CIB         = [32, 0, 32, 0],
;  1507  0             CCB$A_LAST_CIB          = [36, 0, 32, 0],
;  1508  0             CCB$L_STATUS            = [40, 0, 32, 0],
;  1509  0             CCB$A_POOL              = [44, 0, 32, 0],
;  1510  0             CCB$A_FIRST_SPB         = [48, 0, 32, 0],
;  1511  0             CCB$A_LAST_SPB          = [52, 0, 32, 0],
;  1512  0             CCB$V_HASH_BASE         = [56, 0, 0, 0]
;  1513  0         TES;
;  1514  0
;  1515  0     LITERAL
;  1516  0         CCB$K_LOCK_LIST     = BLOCK[0, CCB$A_FIRST_LOCK; , BYTE],
;  1517  0         CCB$K_CLUSTER_LIST  = BLOCK[0, CCB$A_FIRST_CIB; , BYTE],
;  1518  0         CCB$S_HASH_BASE     = BLOCK[0, CCB$V_HASH_BASE; , BYTE],
;  1519  0         CCB$K_SPB_LIST      = BLOCK[0, CCB$A_FIRST_SPB; , BYTE];
;  1520  0
;  1521  0     STRUCTURE
;  1522  0         CCB$HASH[I, O, P, S, E] =
;  1523  0             (CCB$HASH+CCB$S_HASH_BASE+0+(I*HCE$S_BLOCK_LENGTH))<P,S,E>;
;  1524  0
```

```
;   1525  0     LITERAL
;   1526  0         CCB$M_CORRUPT       = 1^1 - 1^0;    ! Stream is corrupt
```

```
;  1527  0     !+
;  1528  0     !       Page Control Block (PCB)
;  1529  0     !
;  1530  0     !
;  1531  0     !        31              15              0
;  1532  0     !       +--------------------------------+ +0
;  1533  0     !       |                                |
;  1534  0     !       |          Page Number           |      PCB$L_PAGE_NUMBER
;  1535  0     !       |                                |
;  1536  0     !       +----------------+---------------+ +4
;  1537  0     !       |                |               |
;  1538  0     !       |  Paging file   |  Page flags   |      PCB$W_FLAGS
;  1539  0     !       |  block number  |               |      PCB$W_BLOCK_NUMBER
;  1540  0     !       +----------------+---------------+ +8
;  1541  0     !       |                                |
;  1542  0     !       |    Pointer to page             |      PCB$A_BUFFER
;  1543  0     !       |  when in virtual memory        |
;  1544  0     !       +--------------------------------+ +12
;  1545  0     !       |                                |
;  1546  0     !       |    Pointer to PCB of           |      PCB$A_NEXT_CLUSTER
;  1547  0     !       |  next page in cluster          |
;  1548  0     !       +--------------------------------+ +16
;  1549  0     !       |                                |
;  1550  0     !       |    Pointer to CIB of           |      PCB$A_PORTAL_CIB
;  1551  0     !       |  cluster's portal page         |
;  1552  0     !       +--------------------------------+ +20
;  1553  0     !       |                                |
;  1554  0     !       |       Ptr to page's FCB        |      PCB$A_FCB
;  1555  0     !       |                                |
;  1556  0     !       +--------------------------------+ +24
;  1557  0     !       |                                |
;  1558  0     !       |     Pointer to page's BCB      |      PCB$A_BCB
;  1559  0     !       |                                |
;  1560  0     !       +--------------------------------+ +28
;  1561  0     !       |                                |
;  1562  0     !       |    Pointer to first            |      PCB$A_FIRST_LCCB
;  1563  0     !       |  LCCB in page cluster          |
;  1564  0     !       +--------------------------------+ +32
;  1565  0     !       |                                |
;  1566  0     !       |    Pointer to last             |      PCB$A_LAST_LCCB
;  1567  0     !       |  LCCB in page cluster          |
;  1568  0     !       +--------------------------------+ +36
;  1569  0     !
;  1570  0     !       A Page Control Block (PCB) exists for every page in the cache,
;  1571  0     !       and for pages that only have presence locks on them (thus they're
;  1572  0     !       not in the staging cache).  Portal pages have a Cluster Information
;  1573  0     !       Block (CIB) appending to their PCB.
;  1574  0     !
;  1575  0     !-
;  1576  0
;  1577  0     LITERAL
;  1578  0         PCB$S_BLOCK_LENGTH = 36;
;  1579  0
;  1580  0     MACRO
;M 1581  0         $PCB = BLOCK[PCB$S_BLOCK_LENGTH,BYTE] FIELD (PCB$Z_FIELDS)
;  1582  0         %;
;  1583  0
```

```
;  1584  0      FIELD   PCB$Z_FIELDS =
;  1585  0          SET
;  1586  0              PCB$L_PAGE_NUMBER       = [0, 0, 32, 0],
;  1587  0              PCB$W_FLAGS             = [4, 0, 16, 0],
;  1588  0              PCB$V_MODIFIED          = [4, 0, 1, 0],     ! Must write back to dict.
;  1589  0              PCB$V_NEW_PAGE          = [4, 3, 1, 0],     ! Page was in free chain
;  1590  0              PCB$V_READ_ONLY         = [4, 4, 1, 0],     ! Page is read only
;  1591  0              PCB$V_FREE_PAGE         = [4, 5, 1, 0],     ! Page is a free page
;  1592  0              PCB$V_AVAILABLE         = [4, 6, 1, 0],     ! Page available in cache
;  1593  0              PCB$V_PORTAL_PAGE       = [4, 7, 1, 0],     ! Page is portal page, CIB follows
;  1594  0              PCB$W_BLOCK_NUMBER      = [6, 0, 16, 0],
;  1595  0              PCB$A_BUFFER            = [8, 0, 32, 0],
;  1596  0              PCB$A_NEXT_CLUSTER      = [12, 0, 32, 0],
;  1597  0              PCB$A_PORTAL_CIB        = [16, 0, 32, 0],
;  1598  0              PCB$A_FCB               = [20, 0, 32, 0],
;  1599  0              PCB$A_BCB               = [24, 0, 32, 0],
;  1600  0              PCB$A_FIRST_LCCB        = [28, 0, 32, 0],
;  1601  0              PCB$A_LAST_LCCB         = [32, 0, 32, 0]
;  1602  0          TES;
;  1603  0
;  1604  0      LITERAL
;  1605  0          PCB$K_LCCB_LIST             = BLOCK[0, PCB$A_FIRST_LCCB; , BYTE];
;  1606  0
;  1607  0      LITERAL
;  1608  0          PCB$M_MODIFIED              = 1^1 - 1^0,    ! Must write back to dict.
;  1609  0          PCB$M_NEW_PAGE              = 1^4 - 1^3,    ! Page was in free chain
;  1610  0          PCB$M_READ_ONLY             = 1^5 - 1^4,    ! Page is read only
;  1611  0          PCB$M_FREE_PAGE             = 1^6 - 1^5,    ! Page is a free page
;  1612  0          PCB$M_AVAILABLE             = 1^7 - 1^6,    ! Page available in cache
;  1613  0          PCB$M_PORTAL_PAGE           = 1^8 - 1^7;    ! Portal page, CIB follows
```

```
;  1614  0     !+
;  1615  0     !       Cluster Information Block (CIB)
;  1616  0     !
;  1617  0     !
;  1618  0     !        31               15               0
;  1619  0     !       +----------------+----------------+ +0
;  1620  0     !       !                !                !
;  1621  0     !       !                !    Cluster     !      CIB$W_FLAGS
;  1622  0     !       !                !     flags      !
;  1623  0     !       +----------------+----------------+ +4   CIB$V_REF_COUNTS
;  1624  0     !       !                !                !
;  1625  0     !       ! Retrieval lock ! Presence lock  !      CIB$W_PRESENCE_COUNT
;  1626  0     !       !   ref count    !   ref count    !      CIB$W_RETRIEVAL_COUNT
;  1627  0     !       +----------------+----------------+ +8
;  1628  0     !       !                !                !
;  1629  0     !       !  Delete lock   !  Update lock   !      CIB$W_UPDATE_COUNT
;  1630  0     !       !   ref count    !   ref count    !      CIB$W_DELETE_COUNT
;  1631  0     !       +----------------+----------------+ +12
;  1632  0     !       !                                 !
;  1633  0     !       !     Pointer to first            !      CIB$A_FIRST_DELETED_CIB
;  1634  0     !       !   deleted child cluster         !
;  1635  0     !       +---------------------------------+ +16
;  1636  0     !       !                                 !
;  1637  0     !       !     Pointer to last             !      CIB$A_LAST_DELETED_CIB
;  1638  0     !       !   deleted child cluster         !
;  1639  0     !       +---------------------------------+ +20
;  1640  0     !       !                                 !
;  1641  0     !       !     Ptr to newest lock          !      CIB$A_LAST_LCB
;  1642  0     !       !    granted for this page        !
;  1643  0     !       +---------------------------------+ +24
;  1644  0     !       !                                 !
;  1645  0     !       !     Ptr to oldest lock          !      CIB$A_FIRST_LCB
;  1646  0     !       !    granted for this page        !
;  1647  0     !       +---------------------------------+ +28
;  1648  0     !       !                                 !
;  1649  0     !       !     Ptr to next cluster         !      CIB$A_NEXT_SIBLING
;  1650  0     !       !   at this directory level       !
;  1651  0     !       +---------------------------------+ +32
;  1652  0     !       !                                 !
;  1653  0     !       !     Ptr to prior cluster        !      CIB$A_PRIOR_SIBLING
;  1654  0     !       !   at this directory level       !
;  1655  0     !       +---------------------------------+ +36
;  1656  0     !       !                                 !
;  1657  0     !       !     Ptr to first cluster        !      CIB$A_FIRST_CHILD
;  1658  0     !       !   at next directory level       !
;  1659  0     !       +---------------------------------+ +40
;  1660  0     !       !                                 !
;  1661  0     !       !     Ptr to last cluster         !      CIB$A_LAST_CHILD
;  1662  0     !       !   at next directory level       !
;  1663  0     !       +---------------------------------+ +44
;  1664  0     !       !                                 !
;  1665  0     !       !     Ptr to CIB in next          !      CIB$A_PARENT_CIB
;  1666  0     !       !    highest directory level      !
;  1667  0     !       +---------------------------------+ +48
;  1668  0     !       !                                 !
;  1669  0     !       !     Ptr to portal CIB           !      CIB$A_HISTORY_CIB
;  1670  0     !       ! in cluster holding history list !
```

```
;    1671  0        !       +--------------------------------+ +52
;    1672  0        !       |                                |
;    1673  0        !       |       Ptr to NCB for the       |      CIB$A_OWNER_NCB
;    1674  0        !       |  node that owns this cluster   |
;    1675  0        !       +--------------------------------+ +56
;    1676  0        !       |                                |
;    1677  0        !       |  Ptr to cluster's pool header  |      CIB$A_POOL
;    1678  0        !       |                                |
;    1679  0        !       +--------------------------------+ +60
;    1680  0        !
;    1681  0        !       Each clusters' portal page has a CIB associated with it.
;    1682  0        !       This block is physically appended to the PCB of the cluster's
;    1683  0        !       portal page.
;    1684  0        !
;    1685  0        !       The CIB also points to the cluster's pool header.
;    1686  0        !
;    1687  0        !       The CIB$V_REF_COUNTS lists must have each of the lock ref counts
;    1688  0        !       in the same order as the LOCK$K_xxx lock request symbols.
;    1689  0        !
;    1690  0        !-
;    1691  0
;    1692  0        LITERAL
;    1693  0            CIB$S_BLOCK_LENGTH = 60 + PCB$S_BLOCK_LENGTH;
;    1694  0
;    1695  0        MACRO
;  M 1696  0            $CIB = BLOCK[CIB$S_BLOCK_LENGTH,BYTE] FIELD (CIB$Z_FIELDS,
;  M 1697  0                                                         PCB$Z_FIELDS)
;    1698  0            %;
;    1699  0
;    1700  0        FIELD   CIB$Z_FIELDS =
;    1701  0            SET
;    1702  0                CIB$W_FLAGS             = [0+PCB$S_BLOCK_LENGTH, 0, 16, 0],
;    1703  0                CIB$V_LOCKED_DELETE     = [0+PCB$S_BLOCK_LENGTH, 0, 1, 0],
;    1704  0                CIB$V_LOCKED_UPDATE     = [0+PCB$S_BLOCK_LENGTH, 1, 1, 0],
;    1705  0                CIB$V_LOCKED_RETRIEVAL  = [0+PCB$S_BLOCK_LENGTH, 2, 1, 0],
;    1706  0                CIB$V_LOCKED_PRESENCE   = [0+PCB$S_BLOCK_LENGTH, 3, 1, 0],
;    1707  0                CIB$V_LOCKED            = [0+PCB$S_BLOCK_LENGTH, 0, 4, 0],
;    1708  0                CIB$V_LOGIN             = [0+PCB$S_BLOCK_LENGTH, 4, 1, 0],
;    1709  0                CIB$V_COMPLETE          = [0+PCB$S_BLOCK_LENGTH, 5, 1, 0],
;    1710  0                CIB$V_NEW               = [0+PCB$S_BLOCK_LENGTH, 6, 1, 0],
;    1711  0                CIB$V_HISTORY           = [0+PCB$S_BLOCK_LENGTH, 7, 1, 0],
;    1712  0                CIB$V_REF_COUNTS        = [4+PCB$S_BLOCK_LENGTH, 0, 0, 0],
;    1713  0                CIB$W_PRESENCE_COUNT    = [4+PCB$S_BLOCK_LENGTH, 0, 16, 0],
;    1714  0                CIB$W_RETRIEVAL_COUNT   = [6+PCB$S_BLOCK_LENGTH, 0, 16, 0],
;    1715  0                CIB$W_UPDATE_COUNT      = [8+PCB$S_BLOCK_LENGTH, 0, 16, 0],
;    1716  0                CIB$W_DELETE_COUNT      = [10+PCB$S_BLOCK_LENGTH, 0, 16, 0],
;    1717  0                CIB$A_FIRST_DELETED_CIB = [12+PCB$S_BLOCK_LENGTH, 0, 32, 0],
;    1718  0                CIB$A_LAST_DELETED_CIB  = [16+PCB$S_BLOCK_LENGTH, 0, 32, 0],
;    1719  0                CIB$A_LAST_LCB          = [20+PCB$S_BLOCK_LENGTH, 0, 32, 0],
;    1720  0                CIB$A_FIRST_LCB         = [24+PCB$S_BLOCK_LENGTH, 0, 32, 0],
;    1721  0                CIB$A_NEXT_SIBLING      = [28+PCB$S_BLOCK_LENGTH, 0, 32, 0],
;    1722  0                CIB$A_PRIOR_SIBLING     = [32+PCB$S_BLOCK_LENGTH, 0, 32, 0],
;    1723  0                CIB$A_FIRST_CHILD       = [36+PCB$S_BLOCK_LENGTH, 0, 32, 0],
;    1724  0                CIB$A_LAST_CHILD        = [40+PCB$S_BLOCK_LENGTH, 0, 32, 0],
;    1725  0                CIB$A_PARENT_CIB        = [44+PCB$S_BLOCK_LENGTH, 0, 32, 0],
;    1726  0                CIB$A_HISTORY_CIB       = [48+PCB$S_BLOCK_LENGTH, 0, 32, 0],
;    1727  0                CIB$A_OWNER_NCB         = [52+PCB$S_BLOCK_LENGTH, 0, 32, 0],
```

```
;    1728  0              CIB$A_POOL              = [56+PCB$S_BLOCK_LENGTH, 0, 32, 0]
;    1729  0          TES;
;    1730  0
;    1731  0      LITERAL
;    1732  0          CIB$K_LOCK_LIST             = BLOCK[0, CIB$A_LAST_LCB; , BYTE],
;    1733  0          CIB$K_SIBLING_LIST          = BLOCK[0, CIB$A_NEXT_SIBLING; , BYTE],
;    1734  0          CIB$K_CHILD_LIST            = BLOCK[0, CIB$A_FIRST_CHILD; , BYTE];
;    1735  0
;    1736  0      LITERAL
;    1737  0          CIB$M_LOCKED_DELETE         = 1^1 - 1^0,
;    1738  0          CIB$M_LOCKED_UPDATE         = 1^2 - 1^1,
;    1739  0          CIB$M_LOCKED_RETRIEVAL      = 1^3 - 1^2,
;    1740  0          CIB$M_LOCKED_PRESENCE       = 1^4 - 1^3,
;    1741  0          CIB$M_LOCKED                = 1^4 - 1^0,
;    1742  0          CIB$M_LOGIN                 = 1^5 - 1^4,    ! Cluster is on login path
;    1743  0          CIB$M_COMPLETE              = 1^6 - 1^5,    ! Cluster is complete in cache
;    1744  0          CIB$M_NEW                   = 1^7 - 1^6,    ! Cluster read in this transaction
;    1745  0          CIB$M_HISTORY               = 1^8 - 1^7;    ! History list is in cluster
```

```
;  1746  0     !+
;  1747  0     !       Lock Control Block (LCB)
;  1748  0     !
;  1749  0     !
;  1750  0     !        31               15               0
;  1751  0     !       +----------------+----------------+ +0
;  1752  0     !       |                |                |
;  1753  0     !       |  Block's flags |   Lock Type    |      LCB$W_LOCK_TYPE
;  1754  0     !       |                |                |      LCB$W_FLAGS
;  1755  0     !       +----------------+----------------+ +4
;  1756  0     !       |                                 |
;  1757  0     !       |       Ptr to prior LCB          |      LCB$A_PRIOR_LCB
;  1758  0     !       |                                 |
;  1759  0     !       +---------------------------------+ +8
;  1760  0     !       |                                 |
;  1761  0     !       |       Ptr to next LCB           |      LCB$A_NEXT_LCB
;  1762  0     !       |                                 |
;  1763  0     !       +---------------------------------+ +12
;  1764  0     !       |                                 |
;  1765  0     !       |       Ptr to locked CIB         |      LCB$A_CIB
;  1766  0     !       |                                 |
;  1767  0     !       +---------------------------------+ +16
;  1768  0     !       |                                 |
;  1769  0     !       |       Ptr to next lock          |      LCB$A_NEXT_LOCK
;  1770  0     !       |    in transaction lock list     |
;  1771  0     !       +---------------------------------+ +20
;  1772  0     !       |                                 |
;  1773  0     !       |       Ptr to prior lock         |      LCB$A_PRIOR_LOCK
;  1774  0     !       |    in transaction lock list     |
;  1775  0     !       +----------------+----------------+ +24
;  1776  0     !
;  1777  0     !       LCBs are used to keep track of which locks exist on a cluster (CIB).
;  1778  0     !
;  1779  0     !       Each LCB is linked to its CIB, and to other LCBs for that cluster.
;  1780  0     !       When an LCB is granted, it is placed in the transaction's
;  1781  0     !       lock list until the transaction terminates.
;  1782  0     !-
;  1783  0
;  1784  0     LITERAL
;  1785  0         LCB$S_BLOCK_LENGTH = 24;
;  1786  0
;  1787  0     MACRO
;M 1788  0         $LCB = BLOCK[LCB$S_BLOCK_LENGTH,BYTE] FIELD (LCB$Z_FIELDS)
;  1789  0         %;
;  1790  0
;  1791  0     FIELD   LCB$Z_FIELDS =
;  1792  0         SET
;  1793  0             LCB$W_LOCK_TYPE         = [0, 0, 16, 0],
;  1794  0             LCB$W_FLAGS             = [2, 0, 16, 0],
;  1795  0             LCB$V_CURRENT           = [2, 1, 1, 0], ! Allocated in current transaction
;  1796  0             LCB$A_PRIOR_LCB         = [4, 0, 32, 0],
;  1797  0             LCB$A_NEXT_LCB          = [8, 0, 32, 0],
;  1798  0             LCB$A_CIB               = [12, 0, 32, 0],
;  1799  0             LCB$A_NEXT_LOCK         = [16, 0, 32, 0],
;  1800  0             LCB$A_PRIOR_LOCK        = [20, 0, 32, 0]
;  1801  0         TES;
;  1802  0
```

```
;  1803  0     LITERAL
;  1804  0         LCB$K_LOCK_LIST             = BLOCK[0, LCB$A_PRIOR_LCB; , BYTE],
;  1805  0         LCB$K_TEMP_LOCK_LIST        = BLOCK[0, LCB$A_NEXT_LOCK; , BYTE];
;  1806  0
;  1807  0     LITERAL
;  1808  0         LCB$M_CURRENT               = 1^2 - 1^1;    ! Allocated in current transaction
```

```
;  1809  0     !+
;  1810  0     !       Location Code Control Block (LCCB)
;  1811  0     !
;  1812  0     !
;  1813  0     !        31              15              0
;  1814  0     !       +--------------------------------+ +0
;  1815  0     !       |                                |
;  1816  0     !       |      Location code number      |      LCCB$L_LOCATION_CODE
;  1817  0     !       |                                |
;  1818  0     !       +--------+--------+--------------+ +4
;  1819  0     !       |        |        |              |      LCCB$W_FLAGS
;  1820  0     !       |  LCCB  |  Block |  LCCB flags  |      LCCB$B_TYPE
;  1821  0     !       |  type  |  type  |              |      LCCB$B_LCCB_TYPE
;  1822  0     !       +--------+--------+--------------+ +8
;  1823  0     !       |                                |
;  1824  0     !       |      Object's disk address     |      LCCB$L_OBJECT/LCCB$V_OBJECT
;  1825  0     !       |                                |
;  1826  0     !       +--------------------------------+ +12
;  1827  0     !       |                                |
;  1828  0     !       |       Pointer to object        |      LCCB$A_OBJECT
;  1829  0     !       |        in virtual memory       |
;  1830  0     !       +--------------------------------+ +16
;  1831  0     !       |                                |
;  1832  0     !       |        Pointer to PCB          |      LCCB$A_PCB
;  1833  0     !       |  for page which holds object   |
;  1834  0     !       +--------------------------------+ +20
;  1835  0     !       |                                |
;  1836  0     !       |       Pointer to next          |      LCCB$A_NEXT_LCCB        ! Virt addr
;  1837  0     !       |       LCCB in cluster          |                              ! must be
;  1838  0     !       +--------------------------------+ +24                         ! 0 mod 4,
;  1839  0     !       |                                |                              ! for queue
;  1840  0     !       |       Pointer to prior         |      LCCB$A_PRIOR_LCCB       ! instructions
;  1841  0     !       |       LCCB in cluster          |
;  1842  0     !       +--------+-----------------------+ +28
;  1843  0     !                |   DPTR of last block  |
;  1844  0     !                | returned by FIND_NEXT |      LCCB$V_LAST_BLOCK
;  1845  0     !                |  (0 if not used yet)  |
;  1846  0     !                +-----------------------+ +31
;  1847  0     !
;  1848  0     !
;  1849  0     !       Location codes are used to provide a convenient means for the
;  1850  0     !       user to identify a particular object.  Each location code is
;  1851  0     !       associated with an LCCB.  The following objects may be
;  1852  0     !       assigned location code:
;  1853  0     !
;  1854  0     !       1)  Entities
;  1855  0     !
;  1856  0     !       2)  Lists
;  1857  0     !
;  1858  0     !       3)  Nodes
;  1859  0     !-
;  1860  0
;  1861  0     LITERAL
;  1862  0         LCCB$S_BLOCK_LENGTH = 31;
;  1863  0
;  1864  0     MACRO
; M 1865  0         $LCCB = BLOCK[LCCB$S_BLOCK_LENGTH,BYTE] FIELD (LCCB$Z_FIELDS)
```

```
;  1866  0          %;
;  1867  0
;  1868  0      FIELD    LCCB$Z_FIELDS =
;  1869  0          SET
;  1870  0              LCCB$L_LOCATION_CODE     = [0, 0, 32, 0],
;  1871  0              LCCB$W_FLAGS             = [4, 0, 16, 0],
;  1872  0              LCCB$V_AVAILABLE         = [4, 0, 1, 0],
;  1873  0              LCCB$V_MUST_SCAN         = [4, 1, 1, 0],
;  1874  0              LCCB$V_GHOST             = [4, 2, 1, 0],
;  1875  0              LCCB$V_DIRECTORY         = [4, 3, 1, 0],
;  1876  0              LCCB$V_TERMINAL          = [4, 4, 1, 0],
;  1877  0              LCCB$V_ENTITY_ATT        = [4, 5, 1, 0],
;  1878  0              LCCB$V_ENTITY_LIST_ATT   = [4, 6, 1, 0],
;  1879  0              LCCB$V_ENTITY_LIST       = [4, 7, 1, 0],
;  1880  0              LCCB$V_STRING_LIST_ATT   = [4, 8, 1, 0],
;  1881  0              LCCB$B_TYPE              = [6, 0, 8, 0],
;  1882  0              LCCB$B_LCCB_TYPE         = [7, 0, 8, 0],
;  1883  0              LCCB$L_OBJECT            = [8, 0, 32, 0],
;  1884  0              LCCB$V_OBJECT            = [8, 0, 24, 0],
;  1885  0              LCCB$A_OBJECT            = [12, 0, 32, 0],
;  1886  0              LCCB$A_PCB               = [16, 0, 32, 0],
;  1887  0              LCCB$A_NEXT_LCCB         = [20, 0, 32, 0],
;  1888  0              LCCB$A_PRIOR_LCCB        = [24, 0, 32, 0],
;  1889  0              LCCB$V_LAST_BLOCK        = [28, 0, 24, 0]
;  1890  0          TES;
;  1891  0
;  1892  0      LITERAL
;  1893  0          LCCB$K_LCCB_LIST             = BLOCK[0, LCCB$A_NEXT_LCCB; , BYTE];
;  1894  0
;  1895  0      LITERAL
;  1896  0          LCCB$M_AVAILABLE             = 1^1 - 1^0,     ! Object's virtual addr is known
;  1897  0          LCCB$M_MUST_SCAN             = 1^2 - 1^1,     ! Protection tree must be scanned
;  1898  0          LCCB$M_GHOST                 = 1^3 - 1^2,     ! LCCB may not be fetched by LCC
;  1899  0          LCCB$M_DIRECTORY             = 1^4 - 1^3,     ! Directory NCB
;  1900  0          LCCB$M_TERMINAL              = 1^5 - 1^4,     ! Terminal NCB
;  1901  0          LCCB$M_ENTITY_ATT            = 1^6 - 1^5,     ! Entity attribute LCCB
;  1902  0          LCCB$M_ENTITY_LIST_ATT       = 1^7 - 1^6,     ! Entity list attribute LCCB
;  1903  0          LCCB$M_ENTITY_LIST           = 1^8 - 1^7,     ! Entity list ECCB
;  1904  0          LCCB$M_STRING_LIST_ATT       = 1^9 - 1^8,     ! String list LCCB
;  1905  0
;  1906  0          LCCB$K_LCCB_TYPE_FIRST       = 1,
;  1907  0          LCCB$K_NCB                   = 1,             ! LCCB includes NCB
;  1908  0          LCCB$K_ECCB                  = 2,             ! LCCB includes ECCB
;  1909  0          LCCB$K_LCCB                  = 3,             ! LCCB
;  1910  0          LCCB$K_LCCB_TYPE_LAST        = 3;
```

```
;    1911  0      !+
;    1912  0      !       Entity Cell Control Block (ECCB)
;    1913  0      !
;    1914  0      !
;    1915  0      !        31              15              0
;    1916  0      !                        +---------------+ +0
;    1917  0      !                        |               |
;    1918  0      !                        |  Cell number  |      ECCB$W_CELL
;    1919  0      !                        |               |
;    1920  0      !                +-------+---------------+ +2
;    1921  0      !                |       |               |
;    1922  0      !                |  DPTR of this cell's  |      ECCB$V_LST
;    1923  0      !                |      LST block        |
;    1924  0      !        +-------+-----------------------+ +5
;    1925  0      !        |       |                       |
;    1926  0      !        |  Ptr to LCCB of the           |      ECCB$A_PARENT_LCCB
;    1927  0      !        |  list that owns this cell     |
;    1928  0      !        +-------------------------------+ +9
;    1929  0      !
;    1930  0      !       Each cell in an entity list may be assigned a location code.
;    1931  0      !
;    1932  0      !       This block is appended to the location code's LCCB to name the
;    1933  0      !       specific cell represented by the location code.
;    1934  0      !-
;    1935  0
;    1936  0
;    1937  0      LITERAL
;    1938  0          ECCB$S_BLOCK_LENGTH =       LCCB$S_BLOCK_LENGTH + 9;
;    1939  0
;    1940  0      MACRO
; M  1941  0          $ECCB = BLOCK[ECCB$S_BLOCK_LENGTH,BYTE] FIELD (ECCB$Z_FIELDS,
; M  1942  0                                                         LCCB$Z_FIELDS)
;    1943  0          %;
;    1944  0
;    1945  0      FIELD   ECCB$Z_FIELDS =
;    1946  0          SET
;    1947  0              ECCB$W_CELL             = [0+LCCB$S_BLOCK_LENGTH, 0, 16, 0],
;    1948  0              ECCB$V_LST              = [2+LCCB$S_BLOCK_LENGTH, 0, 24, 0],
;    1949  0              ECCB$A_PARENT_LCCB      = [5+LCCB$S_BLOCK_LENGTH, 0, 32, 0]
;    1950  0          TES;
```

```
;  1951  0     !+
;  1952  0     !       Node Control Block (NCB)
;  1953  0     !
;  1954  0     !
;  1955  0     !        31              15              0
;  1956  0     !       +---------------+---------------+ +0
;  1957  0     !       |                               |
;  1958  0     !       |      User's access rights     |      NCB$L_ACCESS_RIGHTS
;  1959  0     !       |          to this node         |
;  1960  0     !       +-------------------------------+ +4
;  1961  0     !       |                               |
;  1962  0     !       |      Access rights granted    |      NCB$L_ACCESS_GRANTED
;  1963  0     !       |       by this node's ACL      |
;  1964  0     !       +-------------------------------+ +8
;  1965  0     !       |                               |
;  1966  0     !       |      Access rights denied     |      NCB$L_ACCESS_DENIED
;  1967  0     !       |       by this node's ACL      |
;  1968  0     !       +-------------------------------+ +12
;  1969  0     !       |                               |
;  1970  0     !       |    Access rights banished     |      NCB$L_ACCESS_BANISHED
;  1971  0     !       |       by this node's ACL      |
;  1972  0     !       +-------------------------------+ +16
;  1973  0     !       |                               |
;  1974  0     !       |       Pointer to parent       |      NCB$A_PARENT_NCB
;  1975  0     !       |          NCB block            |
;  1976  0     !       +-------+-------+---------------+ +20  NCB$V_PASSWORD
;  1977  0     !       |Pswrd's|Pswrd's|    Entity's   |      NCB$W_PSW_LENGTH
;  1978  0     !       |class  |data   |    password   |      NCB$B_PSW_DTYPE
;  1979  0     !       |       |type   |     length    |      NCB$B_PSW_CLASS
;  1980  0     !       +-------+-------+---------------+ +24
;  1981  0     !       |                               |
;  1982  0     !       |       Pointer to entity's     |      NCB$A_PSW_POINTER
;  1983  0     !       |         password string       |
;  1984  0     !       +-------------------------------+ +28  NCB$V_NAME
;  1985  0     !       |Name's |Name's |               |      NCB$W_NAME_LENGTH
;  1986  0     !       |class  |data   | Entity name's |      NCB$B_NAME_DTYPE
;  1987  0     !       |       |type   |    length     |      NCB$B_NAME_CLASS
;  1988  0     !       +-------+-------+---------------+ +32
;  1989  0     !       |                               |
;  1990  0     !       |       Pointer to entity's     |      NCB$A_NAME_POINTER
;  1991  0     !       |          name string          |
;  1992  0     !       +-------------------------------+ +36
;  1993  0     !       |                               |
;  1994  0     !       |            Pointer to         |      NCB$A_SPB
;  1995  0     !       |   Security Preservation Block |
;  1996  0     !       +-------------------------------+ +40
;  1997  0     !
;  1998  0     !
;  1999  0     !       The LCCB for a node is followed by an NCB.  This gives additional
;  2000  0     !       information needed for the node.
;  2001  0     !-
;  2002  0
;  2003  0     LITERAL
;  2004  0         NCB$S_BLOCK_LENGTH = 40+LCCB$S_BLOCK_LENGTH;
;  2005  0
;  2006  0     MACRO
;M 2007  0         $NCB = BLOCK[NCB$S_BLOCK_LENGTH,BYTE] FIELD (LCCB$Z_FIELDS,
```

```
;  M 2008  0                                                      NCB$Z_FIELDS)
;    2009  0          %;
;    2010  0
;    2011  0      FIELD    NCB$Z_FIELDS =
;    2012  0          SET
;    2013  0              NCB$L_ACCESS_RIGHTS     = [0+LCCB$S_BLOCK_LENGTH, 0, 32, 0],
;    2014  0              NCB$L_ACCESS_GRANTED    = [4+LCCB$S_BLOCK_LENGTH, 0, 32, 0],
;    2015  0              NCB$L_ACCESS_DENIED     = [8+LCCB$S_BLOCK_LENGTH, 0, 32, 0],
;    2016  0              NCB$L_ACCESS_BANISHED   = [12+LCCB$S_BLOCK_LENGTH, 0, 32, 0],
;    2017  0              NCB$A_PARENT_NCB        = [16+LCCB$S_BLOCK_LENGTH, 0, 32, 0],
;    2018  0              NCB$V_PASSWORD          = [20+LCCB$S_BLOCK_LENGTH, 0, 0, 0],
;    2019  0              NCB$W_PSW_LENGTH        = [20+LCCB$S_BLOCK_LENGTH, 0, 16, 0],
;    2020  0              NCB$B_PSW_DTYPE         = [22+LCCB$S_BLOCK_LENGTH, 0, 8, 0],
;    2021  0              NCB$B_PSW_CLASS         = [23+LCCB$S_BLOCK_LENGTH, 0, 8, 0],
;    2022  0              NCB$A_PSW_POINTER       = [24+LCCB$S_BLOCK_LENGTH, 0, 32, 0],
;    2023  0              NCB$V_NAME              = [28+LCCB$S_BLOCK_LENGTH, 0, 0, 0],
;    2024  0              NCB$W_NAME_LENGTH       = [28+LCCB$S_BLOCK_LENGTH, 0, 16, 0],
;    2025  0              NCB$B_NAME_DTYPE        = [30+LCCB$S_BLOCK_LENGTH, 0, 8, 0],
;    2026  0              NCB$B_NAME_CLASS        = [31+LCCB$S_BLOCK_LENGTH, 0, 8, 0],
;    2027  0              NCB$A_NAME_POINTER      = [32+LCCB$S_BLOCK_LENGTH, 0, 32, 0],
;    2028  0              NCB$A_SPB               = [36+LCCB$S_BLOCK_LENGTH, 0, 32, 0]
;    2029  0          TES;
```

```
;  2030  0     ;+
;  2031  0     ;       File Control Block (FCB)
;  2032  0     ;
;  2033  0     ;
;  2034  0     ;        31               15               0
;  2035  0     ;       +----------------+----------------+ +0
;  2036  0     ;       |                |                |
;  2037  0     ;       |   File's flags |    Channel     |      FCB$W_CHAN
;  2038  0     ;       |                |     number     |      FCB$W_FLAGS
;  2039  0     ;       +-------+--------+----------------+ +4
;  2040  0     ;       |       |                         |
;  2041  0     ;       | File  |    Disk pointer to      |      FCB$V_CDD$TOP
;  2042  0     ;       |Number |    CDD$TOP entity       |      FCB$B_FILE_NUMBER
;  2043  0     ;       +-------+-------------------------+ +8
;  2044  0     ;       |                                 |
;  2045  0     ;       |       First 2 words of          |      FCB$V_FID
;  2046  0     ;       |        the file's FID           |
;  2047  0     ;       +----------------+----------------+ +12
;  2048  0     ;       |                |                |
;  2049  0     ;       | File ID from   | Last word of   |
;  2050  0     ;       | lock manager   |  file's FID    |      FCB$W_LOCK_ID
;  2051  0     ;       +----------------+----------------+ +16
;  2052  0     ;       |                                 |
;  2053  0     ;       |       Pointer to first          |      FCB$A_FREE_PAGE
;  2054  0     ;       |    pre-allocated free page      |
;  2055  0     ;       +-------+--------+----------------+ +20  FCB$V_FILENAME
;  2056  0     ;       |Name's | Name's |    Entity's    |      FCB$W_FILE_LENGTH
;  2057  0     ;       |class  | data   |  file name     |      FCB$B_FILE_DTYPE
;  2058  0     ;       |       | type   |    length      |      FCB$B_FILE_CLASS
;  2059  0     ;       +-------+--------+----------------+ +24
;  2060  0     ;       |                                 |
;  2061  0     ;       |       Pointer to entity's       |      FCB$A_FILE_POINTER
;  2062  0     ;       |         filename string         |
;  2063  0     ;       +---------------------------------+ +28
;  2064  0     ;       |                                 |
;  2065  0     ;       |       DKEY of block which       |      FCB$L_OWNER
;  2066  0     ;       |       points to this file       |
;  2067  0     ;       +-------+--------+----------------+ +32  FCB$V_FULLNAME
;  2068  0     ;       |Name's | Name's |    Entity's    |      FCB$W_FULL_LENGTH
;  2069  0     ;       |class  | data   |  file name     |      FCB$B_FULL_DTYPE
;  2070  0     ;       |       | type   |    length      |      FCB$B_FULL_CLASS
;  2071  0     ;       +-------+--------+----------------+ +36
;  2072  0     ;       |                                 |
;  2073  0     ;       |       Pointer to entity's       |      FCB$A_FULL_POINTER
;  2074  0     ;       |       full filename string      |
;  2075  0     ;       +---------------------------------+ +40
;  2076  0     ;       |                                 |
;  2077  0     ;       |      Pointer to temporary       |      FCB$A_TEMP_FREE
;  2078  0     ;       |         free page list          |
;  2079  0     ;       +---------------------------------+ +44
;  2080  0     ;
;  2081  0     ;       Each active (open) dictionary file has an FCB.
;  2082  0     ;
;  2083  0     ;       The FCB$L_OWNER field holds the DKEY of the attribute block (FIL)
;  2084  0     ;       which pointed to the file.  The primary dictionary file has key
;  2085  0     ;       0, while files that are not currently pointed to have a value of -1
;  2086  0     ;       in the owner field.
```

```
!+

LITERAL
    FCB$K_FILE_LIMIT   = 255,
    FCB$S_BLOCK_LENGTH = 44;

MACRO
    $FCB = BLOCK[FCB$S_BLOCK_LENGTH,BYTE] FIELD (FCB$Z_FIELDS)
    %;

FIELD   FCB$Z_FIELDS =
    SET
        FCB$W_CHAN              = [0, 0, 16, 0],
        FCB$W_FLAGS             = [2, 0, 16, 0],
        FCB$V_READ_ONLY         = [2, 0, 1, 0],
        FCB$V_ROOT              = [2, 1, 1, 0],
        FCB$V_CDD$TOP           = [4, 0, 24, 0],
        FCB$B_FILE_NUMBER       = [7, 0, 8, 0],
        FCB$V_FID               = [8, 0, 0, 0],
        FCB$W_LOCK_ID           = [14, 0, 16, 0],
        FCB$A_FREE_PAGE         = [16, 0, 32, 0],
        FCB$V_FILENAME          = [20, 0, 0, 0],
        FCB$W_FILE_LENGTH       = [20, 0, 16, 0],
        FCB$B_FILE_DTYPE        = [22, 0, 8, 0],
        FCB$B_FILE_CLASS        = [23, 0, 8, 0],
        FCB$A_FILE_POINTER      = [24, 0, 32, 0],
        FCB$L_OWNER             = [28, 0, 32, 0],
        FCB$V_FULLNAME          = [32, 0, 0, 0],
        FCB$W_FULL_LENGTH       = [32, 0, 16, 0],
        FCB$B_FULL_DTYPE        = [34, 0, 8, 0],
        FCB$B_FULL_CLASS        = [35, 0, 8, 0],
        FCB$A_FULL_POINTER      = [36, 0, 32, 0],
        FCB$A_TEMP_FREE         = [40, 0, 32, 0]
    TES;


LITERAL
    FCB$M_READ_ONLY     = 1^1 - 1^0,    ! File can only be opened for read
    FCB$M_ROOT          = 1^2 - 1^1;    ! FCB is the root dictionary file
```

```
;    2126  0      !+
;    2127  0      !       Pre-Allocated Page Block (PAPB)
;    2128  0      !
;    2129  0      !
;    2130  0      !        31               15               0
;    2131  0      !       +---------------------------------+ +0
;    2132  0      !       |                                 |
;    2133  0      !       |      Pointer to next            |      PAPB$A_NEXT
;    2134  0      !       |           BCB                   |
;    2135  0      !       +---------------------------------+ +4
;    2136  0      !       |                                 |
;    2137  0      !       |      Number of free page        |      PAPB$L_NUMBER
;    2138  0      !       |                                 |
;    2139  0      !       +---------------------------------+ +8
;    2140  0      !
;    2141  0      !
;    2142  0      !       Pre-allocated pages are represented by PAPBs linked onto the
;    2143  0      !       FCB in which the pages reside.  Each PAPB has the page number
;    2144  0      !       of the free page it represents.
;    2145  0      !-
;    2146  0
;    2147  0      LITERAL
;    2148  0          PAPB$S_BLOCK_LENGTH = 8;
;    2149  0
;    2150  0      MACRO
;  M 2151  0          $PAPB = BLOCK[PAPB$S_BLOCK_LENGTH,BYTE] FIELD (PAPB$Z_FIELDS)
;    2152  0          %;
;    2153  0
;    2154  0      FIELD   PAPB$Z_FIELDS =
;    2155  0          SET
;    2156  0              PAPB$A_NEXT                     = [0, 0, 32, 0],
;    2157  0              PAPB$L_NUMBER                   = [4, 0, 32, 0]
;    2158  0          TES;
```

```
;  2159  0     !+
;  2160  0     !       Buffer Control Block (BCB)
;  2161  0     !
;  2162  0     !
;  2163  0     !        31               15               0
;  2164  0     !       +---------------------------------+ +0
;  2165  0     !       |                                 |
;  2166  0     !       |       Pointer to next           |      BCB$A_NEXT
;  2167  0     !       |            BCB                  |
;  2168  0     !       +---------------------------------+ +4
;  2169  0     !       |                                 |
;  2170  0     !       |       Pointer to prior          |      BCB$A_PRIOR
;  2171  0     !       |            BCB                  |
;  2172  0     !       +---------------------------------+ +8
;  2173  0     !       |                                 |
;  2174  0     !       |       Pointer to buffer         |      BCB$A_BUFFER
;  2175  0     !       |                                 |
;  2176  0     !       +---------------------------------+ +12
;  2177  0     !       |                                 |
;  2178  0     !       |       Pointer to PCB            |      BCB$A_PCB
;  2179  0     !       |    for page within buffer       |
;  2180  0     !       +---------------------------------+ +16
;  2181  0     !
;  2182  0     !       The buffer control blocks (BCBs) are used to control the pages
;  2183  0     !       that are in memory.
;  2184  0     !
;  2185  0     !       The BCBs are linked into a queue.  When a buffer is needed, the
;  2186  0     !       last buffer in the queue is assigned.
;  2187  0     !
;  2188  0     !       The associated PCB contains information about the page and the
;  2189  0     !       buffer.  If a page is modified while in the buffer, the page
;  2190  0     !       is written back to the work file before the buffer is reused.
;  2191  0     !-
;  2192  0
;  2193  0     LITERAL
;  2194  0         BCB$K_NUMBER        = 16,   ! Number of in-core buffers
;  2195  0         BCB$S_BLOCK_LENGTH  = 16;
;  2196  0
;  2197  0     MACRO
; M 2198  0         $BCB = BLOCK[BCB$S_BLOCK_LENGTH,BYTE] FIELD (BCB$Z_FIELDS)
;  2199  0         %;
;  2200  0
;  2201  0     FIELD  BCB$Z_FIELDS =
;  2202  0         SET
;  2203  0             BCB$A_NEXT              = [0, 0, 32, 0],
;  2204  0             BCB$A_PRIOR             = [4, 0, 32, 0],
;  2205  0             BCB$A_BUFFER            = [8, 0, 32, 0],
;  2206  0             BCB$A_PCB               = [12, 0, 32, 0]
;  2207  0         TES;
;  2208  0
;  2209  0     LITERAL
;  2210  0         BCB$K_BUFFER_LIST           = BLOCK[0, BCB$A_NEXT; , BYTE];
```

```
;  2211  0     !+
;  2212  0     !       User Retrieval List Entry Block
;  2213  0     !       31       23       15       7        0
;  2214  0     !       +--------+--------+--------+--------+ +0
;  2215  0     !       |                   |                 |   URLC$B_COUNT
;  2216  0     !       |      Unused       |      Count      |
;  2217  0     !       |                   |                 |   URLC$V_UNUSED
;  2218  0     !       +--------+--------+--------+--------+ +4
;  2219  0     !
;  2220  0     !
;  2221  0     !
;  2222  0     !       User Retrieval List Entry Block
;  2223  0     !       31       23       15       7        0
;  2224  0     !       +--------+--------+--------+--------+ +0
;  2225  0     !       |                                     |
;  2226  0     !       |          Attribute Name             |   URLE$L_ATT/URLE$L_CELL
;  2227  0     !       |            or                       |
;  2228  0     !       |          Cell Number                |
;  2229  0     !       +--------+--------+--------+--------+ +4
;  2230  0     !       |                   |                 |   URLE$W_DATA_TYPE
;  2231  0     !       |    Reserved       |      Data       |
;  2232  0     !       |                   |    Type         |   URLE$W_RESERVE
;  2233  0     !       |                   |    Expected     |
;  2234  0     !       +--------+--------+--------+--------+ +8
;  2235  0     !       |                                     |
;  2236  0     !       |          Returned Status            |   URLE$L_STATUS
;  2237  0     !       |                                     |
;  2238  0     !       +--------+--------+--------+--------+ +12
;  2239  0     !       |                   |                 |   URLE$W_USER_LENGTH
;  2240  0     !       |                   |    Length       |
;  2241  0     !       | Returned Length   |      of         |   URLE$W_RETURN_LENGTH
;  2242  0     !       |                   |    User Buffer  |
;  2243  0     !       +--------+--------+--------+--------+ +16
;  2244  0     !       |                                     |
;  2245  0     !       |          Address of                 |
;  2246  0     !       |        User Buffer or               |     URLE$A_USER_BUFFER
;  2247  0     !       |       String Descriptor             |
;  2248  0     !       +--------+--------+--------+--------+ +20
;  2249  0
;  2250  0     LITERAL
;  2251  0         URLC$S_BLOCK_LENGTH = 4,
;  2252  0         URLE$S_BLOCK_LENGTH = 20;
;  2253  0
;  2254  0     MACRO
; M 2255  0         $URLC = BLOCK[URLC$S_BLOCK_LENGTH,BYTE] FIELD (URLC$Z_FIELDS)
;   2256  0         %;
;   2257  0
;   2258  0     FIELD   URLC$Z_FIELDS =
;   2259  0         SET
;   2260  0             URLC$B_COUNT    = [0, 0, 8, 0],
;   2261  0             URLC$V_UNUSED   = [1, 0, 24, 0]
;   2262  0         TES;
;   2263  0
;   2264  0
;   2265  0     MACRO
; M 2266  0         $URLE = BLOCK[URLE$S_BLOCK_LENGTH,BYTE] FIELD (URLE$Z_FIELDS)
;   2267  0         %;
;   2268  0
;   2269  0     FIELD   URLE$Z_FIELDS =
;   2270  0         SET
;   2271  0             URLE$L_ATT              = [0, 0, 32, 0],
```

```
;    2268  0              URLE$L_CELL             = [0, 0, 32, 0],
;    2269  0              URLE$W_DATA_TYPE        = [4, 0, 16, 0],
;    2270  0              URLE$W_RESERVE          = [6, 0, 16, 0],
;    2271  0              URLE$L_STATUS           = [8, 0, 32, 0],
;    2272  0              URLE$W_USER_LENGTH      = [12, 0, 16, 0],
;    2273  0              URLE$W_RETURN_LENGTH    = [14, 0, 16, 0],
;    2274  0              URLE$A_USER_BUFFER      = [16, 0, 32, 0]
;    2275  0          TES;
;    2276  0
;    2277  0      STRUCTURE
;    2278  1          URL$BLOCK [I] = (URL$BLOCK + URLC$S_BLOCK_LENGTH +
;    2279  0                          (I * URLE$S_BLOCK_LENGTH));
;    2280  0
```

```
;  2281  0      !+
;  2282  0      !       Write Control Block (WCB)
;  2283  0      !
;  2284  0      !        31               15               0
;  2285  0      !       +----------------+----------------+ +0
;  2286  0      !       |                |                |
;  2287  0      !       |     Flags      |Number of pages |      WCB$W_SIZE
;  2288  0      !       |                |in write clstr  |      WCB$W_FLAGS
;  2289  0      !       +----------------+----------------+ +4
;  2290  0      !       |                                 |
;  2291  0      !       |        Ptr to page's            |      WCB$A_OLD_BUF
;  2292  0      !       |       permanant buffer          |
;  2293  0      !       +---------------------------------+ +8
;  2294  0      !       |                                 |
;  2295  0      !       |        Ptr to page's            |      WCB$A_NEW_BUF
;  2296  0      !       |    location in write buffer     |
;  2297  0      !       +----------------+----------------+ +12  WCB$V_IO_STATUS
;  2298  0      !       |                |                |
;  2299  0      !       |  2nd word of   |  1st word of   |      WCB$W_IO_STATUS
;  2300  0      !       | I/O status blk | I/O status blk |
;  2301  0      !       +----------------+----------------+ +16
;  2302  0      !       |                |                |
;  2303  0      !       |  4th word of   |  3rd word of   |
;  2304  0      !       | I/O status blk | I/O status blk |
;  2305  0      !       +----------------+----------------+ +20
;  2306  0      !
;  2307  0      !
;  2308  0      !-
;  2309  0
;  2310  0      LITERAL
;  2311  0          WCB$S_BLOCK_LENGTH = 20;
;  2312  0
;  2313  0      MACRO
;M 2314  0          $WCB = BLOCK[WCB$S_BLOCK_LENGTH, BYTE] FIELD (WCB$Z_FIELDS)
;  2315  0          %;
;  2316  0
;  2317  0      FIELD   WCB$Z_FIELDS =
;  2318  0          SET
;  2319  0              WCB$W_SIZE              = [0, 0, 16, 0],
;  2320  0              WCB$W_FLAGS             = [2, 0, 16, 0],
;  2321  0              WCB$V_WRITABLE          = [2, 0, 1, 0], ! Write group to file
;  2322  0              WCB$V_GROUP             = [2, 1, 1, 0], ! Group of pages
;  2323  0              WCB$A_OLD_BUF           = [4, 0, 32, 0],
;  2324  0              WCB$A_NEW_BUF           = [8, 0, 32, 0],
;  2325  0              WCB$V_IO_STATUS         = [12, 0, 0, 0],
;  2326  0              WCB$W_IO_STATUS         = [12, 0, 16, 0]
;  2327  0          TES;
;  2328  0
;  2329  0      LITERAL
;  2330  0          WCB$M_WRITABLE      = 1^1 - 1^0,    ! Write group to file
;  2331  0          WCB$M_GROUP         = 1^2 - 1^1;    ! Group of pages
```

```
;    2332  0      !+
;    2333  0      !       Attribute Name Block (ATNM)
;    2334  0      !
;    2335  0      !    31 30 29 28 27 26                  16 15                                0
;    2336  0      !   +--+--+--+--+--+--+--------------------+----------------------------------+
;    2337  0      !   |  |  |  |  |  |  |                    |                                  |
;    2338  0      !   |  |  |  |  |  |  |   Protocol number  |        Attribute number          |
;    2339  0      !   |  |  |  |  |  |  |                    |                                  |
;    2340  0      !   +--+--+--+--+--+--+--------------------+----------------------------------+
;    2341  0      !      |  |  |  |  |  |
;    2342  0      !      |  |  |  |  |  +---  CDD-defined attribute
;    2343  0      !      |  |  |  |  +------  Global attribute
;    2344  0      !      |  |  |  +---------  Customer-defined attribute
;    2345  0      !      |  |  +------------  Reserved
;    2346  0      !      |  +---------------  Reserved
;    2347  0      !      +------------------
;    2348  0      !
;    2349  0      !           The attribute name block defines the break down of the
;    2350  0      !           attribute name.  Bits 0 to 15 contains the number.  Bits 16
;    2351  0      !           to 26 contains the protocol.  The remaining bits are flags
;    2352  0      !           define the type of attribute.  Bit 27 on implies a system
;    2353  0      !           defined attribute.  Bit 28 on implies a global defined attribute.
;    2354  0      !           Bit 29 on implies a customer defined attribute.
;    2355  0      !
;    2356  0      !           NOTE: If the high order word of the attribute name block is 0
;    2357  0      !                 the block is describing a cell. That is the protocol will
;    2358  0      !                 equal 0 and the flag bits will equal 0.
;    2359  0      !-
;    2360  0
;    2361  0      LITERAL
;    2362  0          ATNM$S_BLOCK_LENGTH = 4;
;    2363  0
;    2364  0      MACRO
; M  2365  0          $ATNM = BLOCK[ATNM$S_BLOCK_LENGTH,BYTE] FIELD (ATNM$Z_FIELDS)
;    2366  0          %;
;    2367  0
;    2368  0      FIELD   ATNM$Z_FIELDS =
;    2369  0          SET
;    2370  0              ATNM$W_NUMBER   = [0, 0, 16, 0],
;    2371  0              ATNM$W_HIGH_ORD = [0, 16, 16, 0],
;    2372  0              ATNM$V_PROTOCOL = [0, 16, 11, 0],
;    2373  0              ATNM$V_FLAGS    = [0, 27, 5, 0],
;    2374  0              ATNM$V_SYSTEM   = [0, 27, 1, 0],
;    2375  0              ATNM$V_GLOBAL   = [0, 28, 1, 0],
;    2376  0              ATNM$V_CUSTOMER = [0, 29, 1, 0]
;    2377  0          TES;
```

```
;    2378  0      !+
;    2379  0      !       Page Number Block (PNB)
;    2380  0      !
;    2381  0      !   31                              18 17                                  0
;    2382  0      !   +-------------------------------+-------------------------------------+
;    2383  0      !   |                               |                                     |
;    2384  0      !   |       Page Group Size          |           Page Number               |
;    2385  0      !   |                               |                                     |
;    2386  0      !   +-------------------------------+-------------------------------------+
;    2387  0      !
;    2388  0      !       The page number block define the break down of the page number.
;    2389  0      !       Bits 0 to 17 is the page number.  Bits 18 to 31 define the number
;    2390  0      !       of consecutive pages that can be read or written at one time.
;    2391  0      !-
;    2392  0
;    2393  0      LITERAL
;    2394  0          PNB$S_BLOCK_LENGTH = 4;
;    2395  0
;    2396  0      MACRO
;M   2397  0          $PNB = BLOCK[PNB$S_BLOCK_LENGTH, BYTE] FIELD (PNB$Z_FIELDS)
;    2398  0          %;
;    2399  0
;    2400  0      FIELD   PNB$Z_FIELDS =
;    2401  0          SET
;    2402  0              PNB$V_PAGE_NUM  = [0, 0, 17, 0],
;    2403  0              PNB$V_GROUP_SIZ = [0, 17, 15, 0]
;    2404  0          TES;
```

```
;  2405  0     !+
;  2406  0     !     Security Preservation Block (SPB)
;  2407  0     !
;  2408  0     !      3                   1
;  2409  0     !      1                   5                   0
;  2410  0     !     +--------+--------+--------+--------+ +0
;  2411  0     !     !                                     !
;  2412  0     !     !          Pointer to next            !      SPB$A_NEXT_SPB
;  2413  0     !     !     Securtiy Preservation Block     !
;  2414  0     !     +--------+--------+--------+--------+ +4
;  2415  0     !     !                                     !
;  2416  0     !     !          Pointer to prior           !      SPB$A_PRIOR_SPB
;  2417  0     !     !     Securtiy Preservation Block     !
;  2418  0     !     +--------+--------+--------+--------+ +8
;  2419  0     !     !                                     !
;  2420  0     !     !            Pointer to               !      SPB$A_NCB
;  2421  0     !     !         Node Control Block          !
;  2422  0     !     +--------+--------+--------+--------+ +12
;  2423  0     !     !                                     !
;  2424  0     !     !          Granted Privileges         !      SPB$L_GRANT
;  2425  0     !     !                                     !
;  2426  0     !     +--------+--------+--------+--------+ +16
;  2427  0     !     !                                     !
;  2428  0     !     !          Denied Privileges          !      SPB$L_DENY
;  2429  0     !     !                                     !
;  2430  0     !     +--------+--------+--------+--------+ +20
;  2431  0     !     !                                     !
;  2432  0     !     !          Banished Privileges        !      SPB$L_BANISH
;  2433  0     !     !                                     !
;  2434  0     !     +--------+--------+--------+--------+ +24
;  2435  0     !     !                                     !
;  2436  0     !     !            Access Rights            !      SPB$L_ACCESS_RIGHTS
;  2437  0     !     !                                     !
;  2438  0     !     +--------+--------+--------+--------+ +28    SPB$V_PASSWORD
;  2439  0     !     ! Pswrd's ! Pswrd's !   Entity's      !      SPB$W_PSW_LENGTH
;  2440  0     !     ! class   !  data   !   password      !      SPB$B_PSW_DTYPE
;  2441  0     !     !         !  type   !    length       !      SPB$B_PSW_CLASS
;  2442  0     !     +--------+--------+-----------------+ +32
;  2443  0     !     !                                     !
;  2444  0     !     !       Pointer to entity's           !      SPB$A_PSW_POINTER
;  2445  0     !     !          password string            !
;  2446  0     !     +-------------------------------------+ +36
;  2447  0     !     !                                     !
;  2448  0     !     !                            ! Must   !
;  2449  0     !     !                            ! Scan   !      SPB$B_SCAN_STATUS
;  2450  0     !     !                            ! Status !
;  2451  0     !     !                            +--------+ +37
;  2452  0     !     !
;  2453  0     !     !
;  2454  0     !     !
;  2455  0     !-
;  2456  0
;  2457  0     LITERAL
;  2458  0         SPB$S_BLOCK_LENGTH = 37;
;  2459  0
;  2460  0     MACRO
; M 2461  0         $SPB = BLOCK[SPB$S_BLOCK_LENGTH, BYTE] FIELD (SPB$Z_FIELDS)
```

```
;    2462  0          %;
;    2463  0
;    2464  0      FIELD    SPB$Z_FIELDS =
;    2465  0          SET
;    2466  0              SPB$A_NEXT_SPB          = [0, 0, 32, 0],
;    2467  0              SPB$A_PRIOR_SPB         = [4, 0, 32, 0],
;    2468  0              SPB$A_NCB               = [8, 0, 32, 0],
;    2469  0              SPB$L_GRANT             = [12, 0, 32, 0],
;    2470  0              SPB$L_DENY              = [16, 0, 32, 0],
;    2471  0              SPB$L_BANISH            = [20, 0, 32, 0],
;    2472  0              SPB$L_ACCESS_RIGHTS     = [24, 0, 32, 0],
;    2473  0              SPB$V_PASSWORD          = [28, 0, 0, 0],
;    2474  0              SPB$W_PSW_LENGTH        = [28, 0, 16, 0],
;    2475  0              SPB$B_PSW_DTYPE         = [30, 0, 8, 0],
;    2476  0              SPB$B_PSW_CLASS         = [31, 0, 8, 0],
;    2477  0              SPB$A_PSW_POINTER       = [32, 0, 32, 0],
;    2478  0              SPB$B_SCAN_STATUS       = [36, 0, 8, 0]
;    2479  0          TES;
;    2480  0
;    2481  0      LITERAL
;    2482  0          SPB$K_QUE_HEADER            = BLOCK[0, SPB$A_NEXT_SPB; , BYTE];
```

```
;  2483  0      %SBTTL      'System Literal Definitions'
;  2484  0      !++
;  2485  0      !
;  2486  0      !       SYSTEM LITERAL DEFINITIONS
;  2487  0      !
;  2488  0      !
;  2489  0      !       These literals are only used internally.
;  2490  0      !--
;  2491  0      !
;  2492  0      !+
;  2493  0      !       CDD Implementation Version
;  2494  0      !-
;  2495  0      !
;  2496  0      LITERAL
;  2497  0          CDD$K_FACILITY              = 43,
;  2498  0          CDD$K_LOWEST_VERSION        = 201,  ! Lowest compatible version
;  2499  0          CDD$K_VERSION               = 201;  ! Present version
;  2500  0      !
;  2501  0      !
;  2502  0      !+
;  2503  0      !       Boolean literals
;  2504  0      !-
;  2505  0      !
;  2506  0      LITERAL
;  2507  0          TRUE    = 1,            ! Boolean TRUE value
;  2508  0          FALSE   = 0;            ! Boolean FALSE value
;  2509  0      !
;  2510  0      !
;  2511  0      !+
;  2512  0      !       The following literals are used to validate routines' parameter
;  2513  0      !       lists.
;  2514  0      !-
;  2515  0      !
;  2516  0      LITERAL
;  2517  0          ARG$K_OPTIONAL          = 1,    ! Parameter is optional
;  2518  0          ARG$K_REQUIRED          = 2,    ! Parameter is required
;  2519  0          ARG$K_SYNC              = 3,    ! See CDD$$U_VALIDATE documentation
;  2520  0          ARG$K_SYNCIF            = 4,    ! See CDD$$U_VALIDATE documentation
;  2521  0          ARG$K_MARK              = 0,    ! Mark parameter as missing
;  2522  0          ARG$K_DEFAULT           = 1,    ! Use default value
;  2523  0          ARG$K_STRING            = 1,    ! Default value is a null string
;  2524  0          ARG$K_LONG              = 2,    ! Default value is a longword
;  2525  0          ARG$K_WORD              = 3,    ! Default value is a word value
;  2526  0          ARG$K_REF               = 0,    ! Parameter passed by reference
;  2527  0          ARG$K_VALUE             = 1;    ! Parameter passed by value
;  2528  0      !
;  2529  0      !
;  2530  0      !
;  2531  0      !+
;  2532  0      !       Access Lock Constants
;  2533  0      !
;  2534  0      !       NOTE:   The order of the LOCK$K_NORMAL lock constants MUST be
;  2535  0      !               the same as the order of the CIB$V_REF_COUNTS lock
;  2536  0      !               ref count fields in the CIB.
;  2537  0      !-
;  2538  0      !
;  2539  0      LITERAL
```

```
;    2540  0          LOCK$K_NULL                     = 0,            . Not locked
;    2541  0
;    2542  0          LOCK$K_NORMAL                   = 1,            ! Base of partially queued locks
;    2543  0              LOCK$K_PRESENCE             = 1,
;    2544  0              LOCK$K_RETRIEVAL            = 2,
;    2545  0              LOCK$K_UPDATE               = 3,
;    2546  0              LOCK$K_DELETE               = 4,
;    2547  0          LOCK$K_NORMAL_TYPES             = 4,            ! Number of partially queued locks
;    2548  0
;    2549  0      !!!     LOCK$K_QUEUED               = 5,            ! Base of fully queued locks
;    2550  0      !!!     LOCK$K_FULL_RET             = 5,
;    2551  0      !!!     LOCK$K_FULL_UPD             = 6,
;    2552  0      !!!     LOCK$K_QUEUED_TYPES         = 2,            ! Number of fully queued locks
;    2553  0
;    2554  0          LOCK$K_XXX_GET                  = 1^17-1^16,    ! Flag indicates must establish lock
;    2555  0          LOCK$K_XXX_GET_IF               = 1^18-1^17,    ! Get lock if not already present
;    2556  0          LOCK$K_OPTIONS                  = 1^32-1^16;    ! Option bits for locks
;    2557  0
;    2558  0
;    2559  0      !+
;    2560  0      !       Types of page purge requests
;    2561  0      !-
;    2562  0
;    2563  0      LITERAL
;    2564  0          PURGE$K_ALL             = 1,            ! Complete purge
;    2565  0          PURGE$K_PRESENCE        = 2,            ! Purge but keep portal page
;    2566  0          PURGE$K_RETRIEVAL       = 3,            ! Checkpoint & keep retrieval locks
;    2567  0          PURGE$K_UPDATE          = 4,            ! Checkpoint & keep all locks
;    2568  0          PURGE$K_SUBTREE         = 100,          ! Purge whole subtree
;    2569  0          PURGE$K_CLUSTER         = 101;          ! Purge this cluster only
;    2570  0
;    2571  0
;    2572  0      !+
;    2573  0      !       Types of blocks that can be allocated in a pool.
;    2574  0      !
;    2575  0      !-
;    2576  0
;    2577  0      !+
;    2578  0      !       Types of pools
;    2579  0      !-
;    2580  0
;    2581  0      LITERAL
;    2582  0          MEM$K_LOWEST_POOL   = 1,            ! Lowest pool type
;    2583  0              MEM$K_CCB_POOL  = 1,            ! Pool for CCB and HCBs
;    2584  0              MEM$K_CIB_POOL  = 2,            ! Pool for cluster blocks
;    2585  0          MEM$K_HIGHEST_POOL  = 2;            ! Highest pool type
;    2586  0
;    2587  0
;    2588  0      !+
;    2589  0      !       CCB Pool block types
;    2590  0      !-
;    2591  0
;    2592  0      LITERAL
;    2593  0          MEM$K_CCB_LOWEST            = 1,    ! Lowest block type in CCB pool
;    2594  0              MEM$K_CCB_CCB           = 1,    ! CCB block
;    2595  0              MEM$K_CCB_LOW_SLOT      = 2,    ! Lowest block type in free block list
;    2596  0                  MEM$K_CCB_HCB       = 2,    ! HCB block
```

```
;    2597  0          MEM$K_CCB_HIGHEST           = 2;    ! Highest block type in CCB pool
;    2598  0
;    2599  0
;    2600  0      !+
;    2601  0      !       Cluster Pool block types
;    2602  0      !-
;    2603  0
;    2604  0      LITERAL
;    2605  0          MEM$K_CIB_LOWEST            = 1,    ! Lowest block type in cluster pool
;    2606  0              MEM$K_CIB_CIB           = 1,    ! CIB block
;    2607  0              MEM$K_CIB_LOW_SLOT      = 2,    ! Lowest block type in free block list
;    2608  0                  MEM$K_CIB_PCB       = 2,    ! PCB,
;    2609  0                  MEM$K_CIB_LCB       = 3,    ! LCB,
;    2610  0                  MEM$K_CIB_LCCB      = 4,    ! LCCB,
;    2611  0                  MEM$K_CIB_ECCB      = 5,    ! ECCB,
;    2612  0                  MEM$K_CIB_NCB       = 6,    ! NCB,
;    2613  0                  MEM$K_CIB_SPB       = 7,    ! SPB,
;    2614  0          MEM$K_CIB_HIGHEST           = 7;    ! Highest block type in cluster pool
;    2615  0
;    2616  0      !+
;    2617  0      !       These flags tell the deletion routine how it is to handle the
;    2618  0      !       following cases:
;    2619  0      !
;    2620  0      !       DEL$K_FAST      says that pointers do not have to be cleaned up,
;    2621  0      !                       as the block they reside in is going to be deleted.
;    2622  0      !
;    2623  0      !       DEL$K_PRESERVE  indicates that a directory node is merely to be
;    2624  0      !                       emptied, and that its cluster is not to be deleted.
;    2625  0      !
;    2626  0      !       DEL$K_SUBDICTIONARY     says that sub-files are to have their contents
;    2627  0      !                       deleted.
;    2628  0      !-
;    2629  0
;    2630  0      LITERAL
;    2631  0          DEL$K_FAST          = 1^1 - 1^0,
;    2632  0          DEL$K_PRESERVE      = 1^2 - 1^1,
;    2633  0          DEL$K_SUBDICTIONARY = 1^3 - 1^2;
;    2634  0
;    2635  0
;    2636  0      !+
;    2637  0      !       User Identification Criteria
;    2638  0      !-
;    2639  0
;    2640  0      LITERAL
;    2641  0          CDD$K_ACL_LOWEST    = 1,
;    2642  0          CDD$K_ACL_PASSWORD  = 1,    ! PASSWORD
;    2643  0          CDD$K_ACL_TERMINAL  = 2,    ! TERMINAL name or class
;    2644  0          CDD$K_ACL_UIC       = 3,    ! UIC
;    2645  0          CDD$K_ACL_USERNAME  = 4,    ! USERNAME
;    2646  0          CDD$K_ACL_HIGHEST   = 4;
```

```
;    2647  0        %SBTTL      'Security Masks'
;    2648  0    !++
;    2649  0    !       SECURITY MASKS
;    2650  0    !--
;    2651  0
;    2652  0    !+
;    2653  0    !       CDD security bits
;    2654  0    !-
;    2655  0
;    2656  0    LITERAL
;    2657  0        CDD$K_PROT_C        = 1^1 - 1^0,    ! CONTROL access
;    2658  0        CDD$K_PROT_D        = 1^2 - 1^1,    ! LOCAL DELETE access
;    2659  0        CDD$K_PROT_G        = 1^3 - 1^2,    ! GLOBAL DELETE access
;    2660  0        CDD$K_PROT_H        = 1^4 - 1^3,    ! HISTORY list entry creation access
;    2661  0        CDD$K_PROT_P        = 1^5 - 1^4,    ! PASS THRU access
;    2662  0        CDD$K_PROT_S        = 1^6 - 1^5,    ! SEE (read) access
;    2663  0        CDD$K_PROT_U        = 1^7 - 1^6,    ! UPDATE terminal node access
;    2664  0        CDD$K_PROT_X        = 1^8 - 1^7,    ! EXTEND directory node access
;    2665  0        CDD$K_PROT_F        = 1^9 - 1^8,    ! FORWARDing directory creation allowed
;    2666  0
;    2667  0    !+
;    2668  0    !       Macro-security values
;    2669  0    !-
;    2670  0
;    2671  0        CDD$K_PROT_ANY      = 1^9 - 1^0,
;    2672  0        CDD$K_PROT_DELETE   = CDD$K_PROT_D OR
;    2673  0                              CDD$K_PROT_G,
;    2674  0        CDD$K_PROT_EXTEND   = CDD$K_PROT_F OR
;    2675  0                              CDD$K_PROT_X,
;    2676  0        CDD$K_PROT_UPDATE   = CDD$K_PROT_C OR
;    2677  0                              CDD$K_PROT_D OR
;    2678  0                              CDD$K_PROT_G OR
;    2679  0                              CDD$K_PROT_H OR
;    2680  0                              CDD$K_PROT_U OR
;    2681  0                              CDD$K_PROT_X OR
;    2682  0                              CDD$K_PROT_F,
;    2683  0
;    2684  0    !+
;    2685  0    !       Other processor security bits
;    2686  0    !-
;    2687  0
;    2688  0    !+
;    2689  0    !       VAX-11 Datatrieve
;    2690  0    !-
;    2691  0
;    2692  0        CDD$K_DTR_PROT_E    = 1^17 - 1^16,  ! EXTEND file
;    2693  0        CDD$K_DTR_PROT_R    = 1^18 - 1^17,  ! READ file
;    2694  0        CDD$K_DTR_PROT_M    = 1^19 - 1^18,  ! MODIFY file
;    2695  0        CDD$K_DTR_PROT_W    = 1^20 - 1^19;  ! WRITE file
```

```
;  2696  0        %SBTTL      'User Literal Definitions'
;  2697  0    !+
;  2698  0    !
;  2699  0    !       USER LITERAL DEFINITIONS
;  2700  0    !
;  2701  0    !
;  2702  0    !       These symbols are needed by users of the program interface.
;  2703  0    !-
;  2704  0
;  2705  0
;  2706  0    !+
;  2707  0    !       System Defined Attribute Names
;  2708  0    !-
;  2709  0
;  2710  0    LITERAL
;  2711  0        CDD$K_SYSNAM_FLAGS  = 1^28 OR 1^27 OR 0^16; ! Global/System-defined/Protocol=0
;  2712  0
;  2713  0    LITERAL
;  2714  0        CDD$K_FIRST_SYSNAM  = 1 OR CDD$K_SYSNAM_FLAGS,      ! Lowest system defined attribute name value
;  2715  0        CDD$K_FILE          = 1 OR CDD$K_SYSNAM_FLAGS,      ! Node's file name
;  2716  0        CDD$K_HISTORY       = 2 OR CDD$K_SYSNAM_FLAGS,      ! History list head
;  2717  0        CDD$K_NAME          = 3 OR CDD$K_SYSNAM_FLAGS,      ! Node's name
;  2718  0        CDD$K_PROTOCOL      = 5 OR CDD$K_SYSNAM_FLAGS,      ! Node's protocol name
;  2719  0        CDD$K_TYPE          = 6 OR CDD$K_SYSNAM_FLAGS,      ! Type of object pointed to by location code
;  2720  0        CDD$K_PATHNAME      = 7 OR CDD$K_SYSNAM_FLAGS,      ! Node's complete pathname
;  2721  0        CDD$K_SHORT_PATHNAME= 8 OR CDD$K_SYSNAM_FLAGS,      ! Node's path to CDD$DEFAULT directory
;  2722  0        CDD$K_ORDER         = 9 OR CDD$K_SYSNAM_FLAGS,      ! Directory's order
;  2723  0        CDD$K_LAST_SYSNAM   = 9 OR CDD$K_SYSNAM_FLAGS;      ! Highest system defined attribute name value
;  2724  0
;  2725  0
;  2726  0    !+
;  2727  0    !       Attribute and Entity Types
;  2728  0    !-
;  2729  0
;  2730  0    LITERAL
;  2731  0        CDD$K_FIRST_TYPE    = 1,
;  2732  0        CDD$K_ENTITY        = 1,
;  2733  0        CDD$K_ENTITY_LIST   = 2,
;  2734  0        CDD$K_NULL          = 3,
;  2735  0        CDD$K_NUMERIC       = 4,
;  2736  0        CDD$K_STRING        = 5,
;  2737  0        CDD$K_STRING_LIST   = 6,
;  2738  0        CDD$K_DIRECTORY     = 7,
;  2739  0        CDD$K_TERMINAL      = 8,
;  2740  0        CDD$K_LAST_TYPE     = 8;
;  2741  0
;  2742  0
;  2743  0    !+
;  2744  0    !       User's entity purge options
;  2745  0    !-
;  2746  0
;  2747  0    LITERAL
;  2748  0        CDD$K_ALL           = 1^1 - 1^0,
;  2749  0        CDD$K_ABORT         = 1^2 - 1^1,
;  2750  0        CDD$K_CHECKPOINT    = 1^3 - 1^2;
;  2751  0
;  2752  0
```

```
;  2753  0     !+
;  2754  0     !       User's node creation options
;  2755  0     !-
;  2756  0
;  2757  0     LITERAL
;  2758  0         CDD$K_NOHISTORY     = 1^1 - 1^0,    ! Doesn't want history list cluster
;  2759  0         CDD$K_NOACL         = 1^2 - 1^1,    ! Don't create default ACL entry
;  2760  0         CDD$K_CREATE        = 1^3 - 1^2,    ! Create dictionary file if needed
;  2761  0         CDD$K_FIRST         = 1^4 - 1^3,    ! Insert as first node
;  2762  0         CDD$K_LAST          = 1^5 - 1^4;    ! Insert as last node
;  2763  0
;  2764  0
;  2765  0     !+
;  2766  0     !       User's node deletion options
;  2767  0     !-
;  2768  0
;  2769  0     LITERAL
;  2770  0         CDD$K_CHECK         = 1^1 - 1^0,    ! Fail if directory has children
;  2771  0         CDD$K_SUBDICTIONARY = 1^2 - 1^1;    ! Delete contents of subdictionaries
;  2772  0
;  2773  0
;  2774  0     !+
;  2775  0     !       Values of the CDD$K_ORDER attribute
;  2776  0     !-
;  2777  0
;  2778  0     LITERAL
;  2779  0         CDD$K_SORTED        = 1,            ! Directory is sorted
;  2780  0         CDD$K_NONSORTED     = 2;            ! Directory is not sorted
```

```
;  2781  0          %SBTTL      'LINKAGE DEFINITIONS'
;  2782  0      !++
;  2783  0      !
;  2784  0      !       LINKAGE DEFINITIONS
;  2785  0      !
;  2786  0      !--
;  2787  0
;  2788  0      !+
;  2789  0      !       CDDCALL
;  2790  0      !
;  2791  0      !       This linkage uses the CALLG/CALLS linkage convention, except
;  2792  0      !       that it allows for one global register to be used in
;  2793  0      !       parameter passing.
;  2794  0      !
;  2795  0      !       R11 -- used to pass the user's context pointer.
;  2796  0      !-
;  2797  0
;  2798  0      LINKAGE
;  2799  0          CDDCALL = CALL : GLOBAL (USER_CONTEXT = 11);
;  2800  0
;  2801  0
;  2802  0      !+
;  2803  0      !       SYS_JSB
;  2804  0      !
;  2805  0      !       This linkage provides us with a general JSB routine linkage.
;  2806  0      !-
;  2807  0
;  2808  0      LINKAGE
;  2809  0          SYS_JSB = JSB;
```

```
;    2810  0          %SBTTL      'MACRO DEFINITIONS'
;    2811  0      !++
;    2812  0      !
;    2813  0      !       MACRO DEFINITIONS
;    2814  0      !
;    2815  0      !--
;    2816  0
;    2817  0
;    2818  0      !+
;    2819  0      !       $ACTIVE
;    2820  0      !       $INACTIVE
;    2821  0      !
;    2822  0      !       These macroes declare that we have started, and finished,
;    2823  0      !       respectively, a CDD transaction.  They abort the transaction
;    2824  0      !       if another transaction is in progress.
;    2825  0      !-
;    2826  0
;    2827  0      MACRO
;  M 2828  0          $ACTIVE =
;  M 2829  0              BEGIN
;  M 2830  0                  EXTERNAL
;  M 2831  0                      CDD$GB_INUSE:           BYTE;
;  M 2832  0
;  M 2833  0                  EXTERNAL LITERAL
;  M 2834  0                      CDD$_NOTASTREE;
;  M 2835  0
;  M 2836  0                  BUILTIN
;  M 2837  0                      TESTBITSS;
;  M 2838  0
;  M 2839  0
;  M 2840  0                  IF TESTBITSS (CDD$GB_INUSE) THEN
;  M 2841  0                      SIGNAL (CDD$_NOTASTREE);
;  M 2842  0              END
;    2843  0          %,
;    2844  0
;  M 2845  0          $INACTIVE =
;  M 2846  0              BEGIN
;  M 2847  0                  EXTERNAL
;  M 2848  0                      CDD$GB_INUSE:           BYTE;
;  M 2849  0
;  M 2850  0                  CDD$GB_INUSE = FALSE;
;  M 2851  0              END
;    2852  0          %,
;    2853  0
;    2854  0
;    2855  0      !+
;    2856  0      !       $BITCLEAR
;    2857  0      !
;    2858  0      !       This macro checks to see if any bit in a mask is set in the
;    2859  0      !       target area.  If not, it returns TRUE.
;    2860  0      !-
;    2861  0
;  M 2862  0          $BITCLEAR(target, mask) =
;  M 2863  0              (target AND mask) EQLU 0
;    2864  0          %,
;    2865  0
;    2866  0
```

```
;  2867  0      !+
;  2868  0      !       $BITSET
;  2869  0      !
;  2870  0      !       This macro checks to see if any bit in a mask is set in the
;  2871  0      !       target area.  If so, it returns TRUE.
;  2872  0      !-
;  2873  0
;M 2874  0          $BITSET(target, mask) =
;M 2875  0              (target AND mask) NEQU 0
;  2876  0          %,
;  2877  0
;  2878  0
;  2879  0
;  2880  0      !+
;  2881  0      !       $DONE_TRANS
;  2882  0      !
;  2883  0      !       This macro is used to terminate a transaction.
;  2884  0      !
;  2885  0      !       Call:
;  2886  0      !
;  2887  0      !           $DONE_TRANS [(dsc1 [, dsc2] ...)]
;  2888  0      !
;  2889  0      !       Where:
;  2890  0      !
;  2891  0      !           dsci :== the names of dynamic descriptors which are to
;  2892  0      !                    have their strings returned to the string pool.
;  2893  0      !-
;  2894  0
;M 2895  0          $DONE_TRANS (dsc1) =
;M 2896  0              BEGIN
;M 2897  0                  EXTERNAL ROUTINE
;M 2898  0                      CDD$$SN_DONE_TRANS      : CDDCALL       NOVALUE;
;M 2899  0
;M 2900  0
;M 2901  0                  CDD$$SN_DONE_TRANS (dsc1
;M 2902  0                      %IF NOT %NULL(%REMAINING) %THEN , %REMAINING %FI );
;M 2903  0                  .USER_CONTEXT[CCB$L_STATUS]
;M 2904  0              END
;  2905  0          %,
;  2906  0
;  2907  0
;  2908  0      !+
;  2909  0      !       $FIND_ENTITY
;  2910  0      !
;  2911  0      !       This macro returns the virtual address of the LCCB associated
;  2912  0      !       with a location code.  It also checks to make certain that
;  2913  0      !       the cluster is locked as requested, and that the cluster's node
;  2914  0      !       allows the requested security access.
;  2915  0      !
;  2916  0      !       Call:
;  2917  0      !
;  2918  0      !           lccb-block.wa.v = $FIND_ENTITY (valid-arg.ra.v ,
;  2919  0      !                                           RETRIEVAL           READ
;  2920  0      !               (  CHECK      { UPDATE     } ) , { MODIFY   } );
;  2921  0      !                               DELETE            DELETE
;  2922  0      !                                                 ANY
;  2923  0      !
```

```
;    2924  0       !       Where:
;    2925  0       !
;    2926  0       !           valid-arg :== the address of the calling routine's validated
;    2927  0       !                         argument list.
;    2928  0       !                         The argument list must have the following format:
;    2929  0       !
;    2930  0       !                               valid-arg[0]  ::= address of longword holding
;    2931  0       !                                                 context #
;    2932  0       !                               valid-arg[1]  ::= address of descriptor holding
;    2933  0       !                                                 path name, or zero.
;    2934  0       !
;    2935  0       !           The first set of keywords names the desired lock state of
;    2936  0       !           the entity's cluster.
;    2937  0       !
;    2938  0       !           The last set of keywords names the intended access to the
;    2939  0       !           cluster.
;    2940  0       !-
;    2941  0
;M   2942  0           $FIND_ENTITY (valid_arg, locking, security) =
;M   2943  0               BEGIN
;M   2944  0               EXTERNAL ROUTINE
;M   2945  0                   CDD$$SN_FIND_ENTITY     : CDDCALL;
;M   2946  0
;M   2947  0                   CDD$$SN_FIND_ENTITY (..valid_arg[1],
;M   2948  0                       $FIND_XXX_LOCKING (%REMOVE(locking)),
;M   2949  0                       %NAME('CDD$K_PROT_', security))
;M   2950  0               END
;    2951  0           %,
;    2952  0
;M   2953  0           $FIND_XXX_LOCKING (class, type) =
;M   2954  0               %IF %IDENTICAL (class, %QUOTE LOCK) %THEN
;M   2955  0                   LOCK$K_XXX_GET OR
;M   2956  0               %ELSE
;M   2957  0                   %IF %IDENTICAL (class, %QUOTE LOCKIF) %THEN
;M   2958  0                       LOCK$K_XXX_GET_IF OR
;M   2959  0                   %ELSE
;M   2960  0                       %IF NOT %IDENTICAL (class, %QUOTE CHECK) %THEN
;M   2961  0                           %ERROR ('Invalid locking keyword: ', class)
;M   2962  0                       %FI
;M   2963  0                   %FI
;M   2964  0               %FI
;M   2965  0
;M   2966  0               %NAME ('LOCK$K_', type)
;    2967  0           %,
;    2968  0
;    2969  0
;    2970  0       !+
;    2971  0       !       $FIND_NODE
;    2972  0       !
;    2973  0       !       This macro returns the virtual address of the NCB associated
;    2974  0       !       with a path name or location code.  It also checks to make certain
;    2975  0       !       that the cluster is locked as requested, and that the target node
;    2976  0       !       allows the requested security access.
;    2977  0       !
;    2978  0       !       Call:
;    2979  0       !
;    2980  0       !           ncb-block.wa.v = $FIND_NODE (valid-arg.ra.v ,
```

```
;    2981  0       !                    LOCK          RETRIEVAL          READ
;    2982  0       !                ( { LOCKIF } { UPDATE     } ) , { MODIFY  } );
;    2983  0       !                    CHECK         DELETE             DELETE
;    2984  0       !                                                    ANY
;    2985  0       !
;    2986  0       !       Where:
;    2987  0       !
;    2988  0       !           valid-arg :== the address of the calling routine's validated
;    2989  0       !                         argument list.
;    2990  0       !                         The argument list must have the following format:
;    2991  0       !
;    2992  0       !                               valid-arg[0]  ::= address of longword holding
;    2993  0       !                                                 context #
;    2994  0       !                               valid-arg[1]  ::= address of descriptor holding
;    2995  0       !                                                 path name, or zero.
;    2996  0       !                               valid-arg[2]  ::= address of longword holding
;    2997  0       !                                                 location code, or zero.
;    2998  0       !
;    2999  0       !           The first set of keywords names the desired lock state of
;    3000  0       !           the node's cluster.
;    3001  0       !
;    3002  0       !           The last set of keywords names the intended access to the
;    3003  0       !           cluster.
;    3004  0       !-
;    3005  0
;M   3006  0           $FIND_NODE (valid_arg, locking, security) =
;M   3007  0               BEGIN
;M   3008  0               EXTERNAL ROUTINE
;M   3009  0                   CDD$$SN_FIND_NODE       : CDDCALL;
;M   3010  0
;M   3011  0               CDD$$SN_FIND_NODE (valid_arg,
;M   3012  0                   $FIND_XXX_LOCKING (%REMOVE(locking)),
;M   3013  0                   %NAMET'CDD$K_PROT_', security))
;M   3014  0               END
;    3015  0           %,
;    3016  0
;    3017  0       !+
;    3018  0       !       $FIND_PARENT
;    3019  0       !
;    3020  0       !       This macro returns the virtual address of the NCB associated
;    3021  0       !       with a location code.  It also checks to make certain that
;    3022  0       !       the cluster is locked as requested, and that the cluster's node
;    3023  0       !       allows the requested security access.
;    3024  0       !
;    3025  0       !       Call:
;    3026  0       !
;    3027  0       !           status.wlc.v = $FIND_PARENT (valid-arg.ra.v ,
;    3028  0       !                   LOCK          RETRIEVAL          READ
;    3029  0       !               ( { LOCKIF } { UPDATE     } ) , { MODIFY  } , ncb-block.wa.r,
;    3030  0       !                   CHECK         DELETE             DELETE
;    3031  0       !                                                    ANY
;    3032  0       !               name.wt.ds);
;    3033  0       !
;    3034  0       !       Where:
;    3035  0       !
;    3036  0       !           valid-arg :== the address of the calling routine's validated
;    3037  0       !                         argument list.
```

```
;  3038  0       !                           The argument list must have the following format:
;  3039  0       !
;  3040  0       !                                 valid-arg[0]  ::= address of longword holding
;  3041  0       !                                                   context #
;  3042  0       !                                 valid-arg[1]  ::= address of descriptor holding
;  3043  0       !                                                   path name, or zero.
;  3044  0       !                                 valid-arg[2]  ::= address of longword holding
;  3045  0       !                                                   location code, or zero.
;  3046  0       !
;  3047  0       !           The first set of keywords names the desired lock state of
;  3048  0       !           the target cluster.
;  3049  0       !
;  3050  0       !           The last set of keywords names the intended access to the
;  3051  0       !           cluster.
;  3052  0       !-
;  3053  0
; M 3054  0           $FIND_PARENT (valid_arg, locking, security, ncb_block, name) =
; M 3055  0               BEGIN
; M 3056  0               EXTERNAL ROUTINE
; M 3057  0                   CDD$$SN_FIND_PARENT     : CDDCALL;
; M 3058  0
; M 3059  0               CDD$$SN_FIND_PARENT (valid_arg,
; M 3060  0                   $FIND_XXX_LOCKING (%REMOVE(locking)),
; M 3061  0                   %NAME('CDD$K_PROT_', security), ncb_block, name)
; M 3062  0               END
;  3063  0           %,
;  3064  0
;  3065  0       !+
;  3066  0       !       $INIT_DSC
;  3067  0       !
;  3068  0       !       This macro is used to initialize dynamic string descriptors.
;  3069  0       !
;  3070  0       !
;  3071  0       !       Call:
;  3072  0       !
;  3073  0       !           $INIT_DSC (dsc1 [, dsc2] ...)
;  3074  0       !
;  3075  0       !-
;  3076  0
; M 3077  0           $INIT_DSC[DSC_NAM] =
; M 3078  0               BEGIN
; M 3079  0               DSC_NAM[DSC$B_DTYPE] = DSC$K_DTYPE_T;
; M 3080  0               DSC_NAM[DSC$B_CLASS] = DSC$K_CLASS_D;
; M 3081  0               DSC_NAM[DSC$W_LENGTH] = 0;
; M 3082  0               DSC_NAM[DSC$A_POINTER] = 0;
; M 3083  0               END
;  3084  0           %,
;  3085  0
;  3086  0
;  3087  0
;  3088  0       !+
;  3089  0       !       $IO_SYNC (ef, iosb)
;  3090  0       !
;  3091  0       !       This macro waits for the event flag (ef) to be set and for
;  3092  0       !       the I/O status parameter (iosb) to be filled in (non-zero).
;  3093  0       !
;  3094  0       !       ef      must be the target event flag number.
```

```
;    3095  0      !
;    3096  0      !       iosb    must be the address of the I/O status block.  This
;    3097  0      !               must be defined as a VECTOR[,WORD] VOLATILE structure.
;    3098  0      !-
;    3099  0
; M  3100  0          $IO_SYNC (ef, iosb) =
; M  3101  0              DO
; M  3102  0                  BEGIN
; M  3103  0                      LOCAL
; M  3104  0                          STATUS:             LONG;
; M  3105  0
; M  3106  0                      STATUS = $WAITFR (efn = ef);
; M  3107  0                      IF NOT .STATUS THEN
; M  3108  0                          SIGNAL_STOP (.STATUS);
; M  3109  0                      STATUS = $CLREF (efn = ef);
; M  3110  0                      IF NOT .STATUS THEN
; M  3111  0                          SIGNAL_STOP (.STATUS);
; M  3112  0                  END WHILE .iosb[0] EQLU 0
;    3113  0          %,
;    3114  0
;    3115  0
;    3116  0
;    3117  0      !+
;    3118  0      !       $MARK_PAGE (page-block)
;    3119  0      !
;    3120  0      !       This macro marks a page as modified.  Such a page must be
;    3121  0      !       written back to the dictionary file when it is purged from
;    3122  0      !       the staging buffers.
;    3123  0      !-
;    3124  0
; M  3125  0          $MARK_PAGE (page_block) =
; M  3126  0              page_block[PCB$V_MODIFIED] = TRUE
;    3127  0          %,
;    3128  0
;    3129  0
;    3130  0      !+
;    3131  0      !       $PARAMETERS (arg1 [, arg2] ...)
;    3132  0      !
;    3133  0      !       This macro is used to build the control vector for the parameter
;    3134  0      !       list validate routine (CDD$$U_VALIDATE).
;    3135  0      !
;    3136  0      !       There is one entry (arg1, arg2, etc) for every formal parameter in
;    3137  0      !       the routine.  Each entry has the following format:
;    3138  0      !
;    3139  0      !       ( REQUIRED , [n] , [REF ! VALUE] !
;    3140  0      !         { SYNC ! SYNCIF }, n    !
;    3141  0      !         OPTIONAL [,MARK ! ,DEFAULT [,STRING ! ,LONG ! ,WORD] ] )
;    3142  0      !-
;    3143  0
; M  3144  0          $PARAMETERS[] =
; M  3145  0              UPLIT (%LENGTH, $PARM_PROCESS (%REMOVE(%REMAINING)))
;    3146  0          %,
;    3147  0
; M  3148  0          $PARM_PROCESS[arg] =
; M  3149  0              BYTE ($PARM_DECIDE (%REMOVE(arg)))
;    3150  0          %,
;    3151  0
```

```
;M 3152  0          $PARM_DECIDE(status)[] =
;M 3153  0              %IF %IDENTICAL (status, %QUOTE REQUIRED) %THEN
;M 3154  0                  ARG$K_REQUIRED, 0,
;M 3155  0                  %IF %NULL (%REMAINING) %THEN
;M 3156  0                      ARG$K_REF, 0
;M 3157  0                  %ELSE
;M 3158  0                      $PARM_REQUIRED (%REMAINING)
;M 3159  0                  %FI
;M 3160  0              %ELSE
;M 3161  0                  %IF %IDENTICAL (status, %QUOTE SYNC) %THEN
;M 3162  0                      ARG$K_SYNC, 0, 0, %REMAINING
;M 3163  0                  %ELSE
;M 3164  0                      %IF %IDENTICAL (status, %QUOTE SYNCIF) %THEN
;M 3165  0                          ARG$K_SYNCIF, 0, 0, %REMAINING
;M 3166  0                      %ELSE
;M 3167  0                          %IF %IDENTICAL (status, %QUOTE OPTIONAL) %THEN
;M 3168  0                              ARG$K_OPTIONAL, $PARM_OPTIONAL (%REMAINING), 0
;M 3169  0                          %ELSE
;M 3170  0                              %ERROR ('Invalid parameter status: ', status)
;M 3171  0                              0, 0, 0, 0
;M 3172  0                          %FI
;M 3173  0                      %FI
;M 3174  0                  %FI
;M 3175  0              %FI
;   3176  0          %,
;   3177  0
;M 3178  0          $PARM_REQUIRED(number, pass) =
;M 3179  0              %IF %NULL (pass) %THEN
;M 3180  0                  0
;M 3181  0              %ELSE
;M 3182  0                  %NAME ('ARG$K_', pass)
;M 3183  0              %FI
;M 3184  0              %IF %NULL (number) %THEN
;M 3185  0                  , 0
;M 3186  0              %ELSE
;M 3187  0                  , number
;M 3188  0              %FI
;   3189  0          %,
;   3190  0
;M 3191  0          $PARM_OPTIONAL(action)[] =
;M 3192  0              %IF %IDENTICAL (action, %QUOTE MARK) %THEN
;M 3193  0                  ARG$K_MARK, 0
;M 3194  0              %ELSE
;M 3195  0                  %IF %IDENTICAL (action, %QUOTE DEFAULT) %THEN
;M 3196  0                      ARG$K_DEFAULT, %NAME ('ARG$K_', %REMAINING)
;M 3197  0                  %ELSE
;M 3198  0                      %ERROR ('Invalid action keyword: ', action)
;M 3199  0                      0, 0
;M 3200  0                  %FI
;M 3201  0              %FI
;   3202  0          %,
;   3203  0
;   3204  0
;   3205  0      !+
;   3206  0      !       $PRESENT
;   3207  0      !
;   3208  0      !       This macro checks to see if a parameter is present (non-zero)
```

```
;    3209  0       !       in a list.
;    3210  0       !-
;    3211  0
; M  3212  0           $PRESENT (name) =
; M  3213  0               .name NEQA 0
;    3214  0           %,
;    3215  0
;    3216  0
;    3217  0       !+
;    3218  0       !       $RECOVERY ( RESET ! ENABLE ! DISABLE )
;    3219  0       !
;    3220  0       !       This macro determines the ability of the exit handler to
;    3221  0       !       perform a purge of the cache if the task aborts.
;    3222  0       !
;    3223  0       !       $RECOVERY (DISABLE)
;    3224  0       !                       declares that the internal data structures or
;    3225  0       !                       external disk structure is in an indeterminant
;    3226  0       !                       state and cannot be recovered by the exit
;    3227  0       !                       handler.
;    3228  0       !
;    3229  0       !       $RECOVERY (ENABLE)
;    3230  0       !                       declares that the data structure manipulation
;    3231  0       !                       is completed.
;    3232  0       !
;    3233  0       !       $RECOVERY (RESET)
;    3234  0       !                       specifies that the data structures are in a
;    3235  0       !                       recoverable state.
;    3236  0       !
;    3237  0       !       Note that these can be nested.  Recovery is only possible if
;    3238  0       !       the recovery counter is zero (reset).
;    3239  0       !-
;    3240  0
; M  3241  0           $RECOVERY (option) =
; M  3242  0               BEGIN
; M  3243  0                   EXTERNAL
; M  3244  0                       CDD$GW_RECOVERY:        WORD;
; M  3245  0
; M  3246  0                   %IF %IDENTICAL (option, %QUOTE DISABLE) %THEN
; M  3247  0                       CDD$GW_RECOVERY = .CDD$GW_RECOVERY + 1;
; M  3248  0                   %ELSE
; M  3249  0                       %IF %IDENTICAL (option, %QUOTE ENABLE) %THEN
; M  3250  0                           CDD$GW_RECOVERY = .CDD$GW_RECOVERY - 1;
; M  3251  0                       %ELSE
; M  3252  0                           %IF %IDENTICAL (option, %QUOTE RESET) %THEN
; M  3253  0                               CDD$GW_RECOVERY = 0;
; M  3254  0                           %ELSE
; M  3255  0                               %ERROR ('Illegal recovery option: ', option)
; M  3256  0                           %FI
; M  3257  0                       %FI
; M  3258  0                   %FI
; M  3259  0               END
;    3260  0           %,
;    3261  0
;    3262  0
;    3263  0       !
;    3264  0       !+
;    3265  0       !       $RELEASE_LOCK
```

```
;  3266  0       !
;  3267  0       !       This macro calls the CDD$$SN_RELEASE routine to release one
;  3268  0       !       or more locks.
;  3269  0       !
;  3270  0       !       Call:
;  3271  0       !
;  3272  0       !                               RETRIEVAL
;  3273  0       !           $RELEASE_LOCK ( { UPDATE } , ncb-block1 ...);
;  3274  0       !                               DELETE
;  3275  0       !-
;  3276  0
;M 3277  0           $RELEASE_LOCK (locking)[] =
;M 3278  0               BEGIN
;M 3279  0               EXTERNAL ROUTINE
;M 3280  0                   CDD$$SN_RELEASE          : CDDCALL       NOVALUE;
;M 3281  0
;M 3282  0               CDD$$SN_RELEASE (%NAME ('LOCK$K_', locking), %REMAINING)
;M 3283  0               END
;  3284  0           %,
;  3285  0
;  3286  0
;  3287  0       !+
;  3288  0       !       $SIGNAL_SEVERE (error)
;  3289  0       !
;  3290  0       !       This routine signals a severe error.
;  3291  0       !-
;  3292  0
;M 3293  0           $SIGNAL_SEVERE (ERROR) =
;M 3294  0               SIGNAL (ERROR OR S'S$K_SEVERE)
;  3295  0           %,
;  3296  0
;  3297  0
;  3298  0       !+
;  3299  0       !       $STATIC_DSC
;  3300  0       !
;  3301  0       !       This macro is used to initialize static string descriptors.
;  3302  0       !
;  3303  0       !
;  3304  0       !       Call:
;  3305  0       !
;  3306  0       !           $STATIC_DSC (dsc1 [, dsc2] ...)
;  3307  0       !
;  3308  0       !       Where:
;  3309  0       !
;  3310  0       !           dsci :== name ! ( name , source )
;  3311  0       !
;  3312  0       !       Source is the name of another descriptor.  The named descriptor
;  3313  0       !       is initialized to point to the same string as source.
;  3314  0       !
;  3315  0       !-
;  3316  0
;M 3317  0           $STATIC_DSC[dsci] =
;M 3318  0               $STATIC_DSC_BUILD (%REMOVE (dsci))
;  3319  0           %,
;  3320  0
;M 3321  0           $STATIC_DSC_BUILD(name, source) =
;M 3322  0               BEGIN
```

```
;  M 3323  0                name[DSC$B_DTYPE] = DSC$K_DTYPE_T;
;  M 3324  0                name[DSC$B_CLASS] = DSC$K_CLASS_S;
;  M 3325  0                %IF %NULL(source) %THEN
;  M 3326  0                    name[DSC$W_LENGTH] = 0;
;  M 3327  0                    name[DSC$A_POINTER] = 0;
;  M 3328  0                %ELSE
;  M 3329  0                    name[DSC$W_LENGTH] = .source[DSC$W_LENGTH];
;  M 3330  0                    name[DSC$A_POINTER] = .source[DSC$A_POINTER];
;  M 3331  0                %FI
;  M 3332  0            END
;    3333  0        %,
;    3334  0
;    3335  0
;    3336  0        !+
;    3337  0        !       $STRING
;    3338  0        !
;    3339  0        !       These macros are used to build string descriptors for literal
;    3340  0        !       strings.
;    3341  0        !
;    3342  0        !       Call:
;    3343  0        !
;    3344  0        !               $STRING ( (name, string) ...)
;    3345  0        !               $STRING_INIT ();
;    3346  0        !
;    3347  0        !       "name" is defined to be a BLOCK structure, and the address of
;    3348  0        !       the string is poked into the structure by the STRING_INIT macro,
;    3349  0        !       which must be the first executable statement in a routine.
;    3350  0        !-
;    3351  0
;  M 3352  0        $STRING [] =
;  M 3353  0            $STRING_DSC_SETUP (%REMAINING)
;  M 3354  0            MACRO
;  M 3355  0                %QUOTE %QUOTE $STRING_INIT =
;  M 3356  0                    $STRING_PTR_SETUP (%QUOTE %EXPAND %REMAINING)
;  M 3357  0                %QUOTE %
;    3358  0        %,
;    3359  0
;  M 3360  0        $STRING_DSC_SETUP[PAIR] =
;  M 3361  0            $STRING_DSC_INIT (%REMOVE(PAIR))
;    3362  0        %,
;    3363  0
;  M 3364  0        $STRING_DSC_INIT (STR_NAME, STR_VAL) =
;  M 3365  0            OWN
;  M 3366  0                STR_NAME:   $DSC PRESET ([DSC$B_DTYPE] = DSC$K_DTYPE_T,
;  M 3367  0                                         [DSC$B_CLASS] = DSC$K_CLASS_S,
;  M 3368  0                                         [DSC$W_LENGTH] =
;  M 3369  0                                             %CHARCOUNT (%REMOVE(STR_VAL)));
;    3370  0        %,
;    3371  0
;  M 3372  0        $STRING_PTR_SETUP[PAIR] =
;  M 3373  0            $STRING_PTR_INIT (%REMOVE(PAIR))
;    3374  0        %,
;    3375  0
;  M 3376  0        $STRING_PTR_INIT (STR_NAME, STR_VAL) =
;  M 3377  0            STR_NAME[DSC$A_POINTER] = UPLIT BYTE(%REMOVE(STR_VAL));
;    3378  0        %,
;    3379  0
```

```
;    3380  0
;    3381  0
;    3382  0       !+
;    3383  0       !       $TEXTC
;    3384  0       !
;    3385  0       !       This macro is used to define counted strings.  The first byte
;    3386  0       !       of such strings is a count of the number of characters in the
;    3387  0       !       string.
;    3388  0       !
;    3389  0       !       Each string is defined to be a VECTOR[,BYTE] structure, with
;    3390  0       !       the 0 element being the character count, and the actual string
;    3391  0       !       starting at STRING[1].
;    3392  0       !
;    3393  0       !       $TEXTC ((name1,'str1') [, (name2, 'str2')] ...);
;    3394  0       !
;    3395  0       !-
;    3396  0
;  M 3397  0           $TEXTC[PAIR] =
;  M 3398  0               $TEXTC_STR(%REMOVE(PAIR))
;    3399  0           %,
;    3400  0
;  M 3401  0           $TEXTC_STR(NAME,TSTR) =
;  M 3402  0               BIND
;  M 3403  0                   NAME = UPLIT BYTE (%CHARCOUNT(%REMOVE(TSTR)), %REMOVE(TSTR)) :
;  M 3404  0                       VECTOR[%CHARCOUNT(%REMOVE(TSTR))+1, BYTE];
;    3405  0           %,
;    3406  0
;    3407  0
;    3408  0       !+
;    3409  0       !       $VALIDATE (cntl, [arg-list])
;    3410  0       !
;    3411  0       !       This macro generates a call to the general transaction setup
;    3412  0       !       routine.
;    3413  0       !
;    3414  0       !       cntl    -- the address of the control vector for CDD$$U_VALIDATE.
;    3415  0       !
;    3416  0       !       arg-list-- is optional.  If present, it is the address of the
;    3417  0       !                  vector which is to receive the verified argument list
;    3418  0       !                  from CDD$$U_VALIDATE.
;    3419  0       !-
;    3420  0
;  M 3421  0           $VALIDATE(cntl, arg_list) =
;  M 3422  0               BEGIN
;  M 3423  0                   EXTERNAL ROUTINE
;  M 3424  0                       CDD$$SN_START_TRANS     : CDDCALL NOVALUE;
;  M 3425  0
;  M 3426  0                   BUILTIN
;  M 3427  0                       AP;
;  M 3428  0
;  M 3429  0
;  M 3430  0                   %IF %NULL (arg_list) %THEN
;  M 3431  0                       CDD$$SN_START_TRANS (.AP, cntl)
;  M 3432  0                   %ELSE
;  M 3433  0                       CDD$$SN_START_TRANS (.AP, cntl, arg_list)
;  M 3434  0                   %FI
;  M 3435  0               END
;    3436  0           %;
```

```
;                                       COMMAND QUALIFIERS

;       BLISS/LIST=LIS$:CDDLIB/LIBRARY=OBJ$:CDDLIB SRC$:CDDLIB

; Run Time:         00:15.0
; Elapsed Time:     00:59.0
; Lines/CPU Min:    13734
; Lexemes/CPU-Min: 38142
; Memory Used:  144 pages
; Library Precompilation Complete
```

CDDENTRY
LIS

CDD

CDDSHR
MAP

CDDEXC2
LIS

CDDLIB
LIS

CDDEXC
LIS

WRITEBOOT
LIS

CDDLIB
B32

GENRALREQ
R32

LEXICAL
LIS

GENCODE4
LIS

CLI5DEF
R32

CDUMSGS
LIS

CDU

CDU
MAP

GENCODE1
LIS

CDUREQ
R02

GENCODE3
LIS

GENCODE2
LIS

CDDTYPDEF
LIS