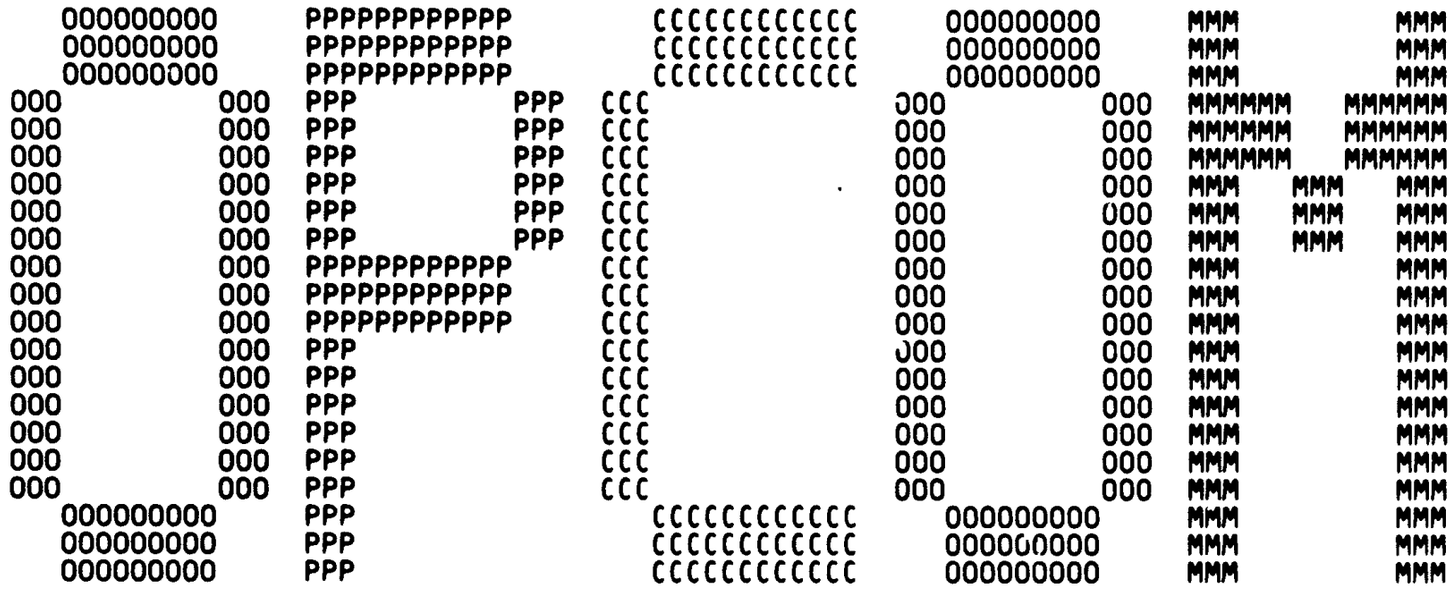


_$2

Sym

ALL

ASC

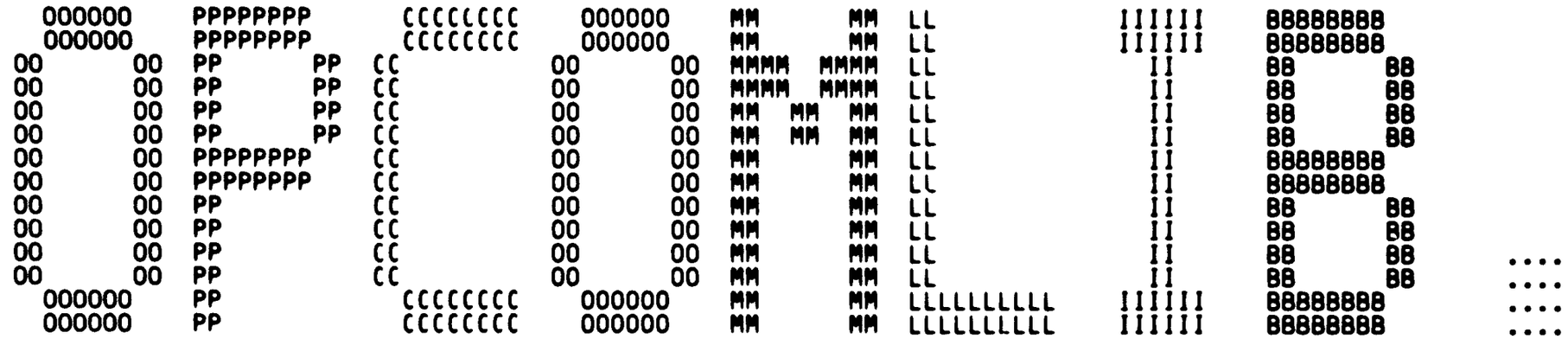
BOD
BOD
BOD
BOD
BOD
BOD
BOD
BOD
BUG
BYP
CAN
CAN
CAN
CHE
CHE

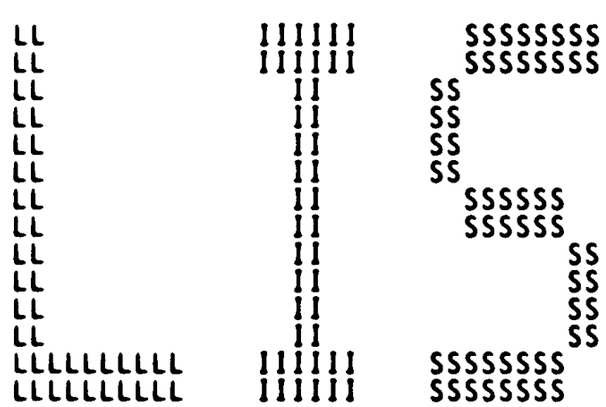
CLU
CLU
CLU
CLU
CLU
CLU
CLU

CLU
CLU
CLU
CLU
CLU
CLU
CLU
CLU

CLU
CLU

FILEID**OPCOMLIB





```
0001  0     !MODULE OPC$OPCOMLIB    (IDENT = 'V04-000') =   %TITLE 'Facility-wide library module'
0002  0     ! BEGIN
0003  0     !
0004  0     !*****************************************************************************
0005  0     !*                                                                           *
0006  0     !*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                  *
0007  0     !*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                   *
0008  0     !*  ALL RIGHTS RESERVED.                                                     *
0009  0     !*                                                                           *
0010  0     !*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED    *
0011  0     !*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE   *
0012  0     !*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER    *
0013  0     !*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY    *
0014  0     !*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY    *
0015  0     !*  TRANSFERRED.                                                             *
0016  0     !*                                                                           *
0017  0     !*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE    *
0018  0     !*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT    *
0019  0     !*  CORPORATION.                                                             *
0020  0     !*                                                                           *
0021  0     !*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS    *
0022  0     !*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                  *
0023  0     !*                                                                           *
0024  0     !*                                                                           *
0025  0     !*****************************************************************************
0026  0     !
0027  0     !++
0028  0     ! FACILITY:     OPCOM - Operator communication facility
0029  0     !
0030  0     ! ABSTRACT:     BLISS Library for OPCOM facility
0031  0     !
0032  0     ! ENVIRONMENT:  VAX/VMS User mode
0033  0     !
0034  0     ! AUTHOR:       CW Hobbs
0035  0     !
0036  0     ! CREATED:      24-Aug-1983
0037  0     !
0038  0     ! MODIFIED BY:
0039  0     !
0040  0     !       V03-003 CWH3003         CW Hobbs                    12-Aug-1984
0041  0     !               Remove REPLY and SOFTWARE operators from the known list
0042  0     !
0043  0     !       V03-002 CWH3002         CW Hobbs                    16-Sep-1983
0044  0     !               Move non-facility EXTERNAL ROUTINE declarations to this file
0045  0     !
0046  0     !--
0047  0     !
0048  0     !
0049  0     ! Include the data structure definitions from external files
0050  0     !
0051  0     LIBRARY 'SYS$LIBRARY:LIB';
0052  0     REQUIRE 'LIB$:OPCDEFTMP';       ! New message definitions
0293  0     REQUIRE 'LIB$:OPCOMDEF';
0935  0     REQUIRE 'SHRLIB$:CLUMBX';
```

```
;  0998  0     !
;  0999  0     ! Define a compile time variable to be used as a table origin by several of the macros.
;  1000  0     !
;  1001  0     COMPILETIME     counter = 0;
;  1002  0
;  1003  0     !
;  1004  0     ! Define literal values
;  1005  0     !
;  1006  0     LITERAL
;  1007  0             max_dev_nam     = 64,                   ! Maximum length of a dev name
;  1008  0             !
;  1009  0             ! Define volume protection masks that are used to control access to the operator mailbox.
;  1010  0             !
;  1011  0             read_write      = %X'0FF0F',            ! Allow read and write access
;  1012  0             read_nowrite    = %X'0FFFF',            ! Allow read, don't allow write
;  1013  0             !
;  1014  0             ! Other useful literals
;  1015  0             !
;  1016  0             true            = 1,                    ! BOOLEAN value
;  1017  0             false           = 0,                    ! BOOLEAN value
;  1018  0             !
;  1019  0             on              = 1,                    ! Bit value
;  1020  0             off             = 0,                    ! Bit value
;  1021  0             !
;  1022  0             ! Define the masks containing all known operators
;  1023  0             !
;  1024  1             known_attn_mask1 =      (OPC$M_NM_CENTRL OR     OPC$M_NM_PRINT  OR      OPC$M_NM_TAPES  OR
;  1025  1                                      OPC$M_NM_DISKS  OR      OPC$M_NM_DEVICE OR      OPC$M_NM_CARDS  OR
;  1026  1                                      OPC$M_NM_NTWORK OR      OPC$M_NM_CLUSTER OR     OPC$M_NM_SECURITY OR
;  1027  1                                     !OPC$M_NM_REPLY  OR      OPC$M_NM_SOFTWARE OR    OPC$M_NM_FILL_11 OR
;  1028  1                                      OPC$M_NM_OPER1  OR      OPC$M_NM_OPER2  OR      OPC$M_NM_OPER3  OR
;  1029  1                                      OPC$M_NM_OPER4  OR      OPC$M_NM_OPER5  OR      OPC$M_NM_OPER6  OR
;  1030  1                                      OPC$M_NM_OPER7  OR      OPC$M_NM_OPER8  OR      OPC$M_NM_OPER9  OR
;  1031  0                                      OPC$M_NM_OPER10 OR      OPC$M_NM_OPER11 OR      OPC$M_NM_OPER12),
;  1032  0             known_attn_mask2 =      0;
;  1033  0
;  1034  0     !
;  1035  0     ! Define routine linkages
;  1036  0     !
;  1037  0     LINKAGE
;  1038  0             alloc_csd       = JSB (REGISTER=1; REGISTER=2)
;  1039  0                                     : NOPRESERVE (3) NOTUSED (4,5,6,7,8,9,10,11),
;  1040  0             csp_call        = JSB (REGISTER=2)
;  1041  0                                     : NOTUSED (3,4,5,6,7,8,9,10,11),
;  1042  0             dalloc_csd      = JSB (REGISTER=0)
;  1043  0                                     : NOPRESERVE (2,3) NOTUSED (4,5,6,7,8,9,10,11),
;  1044  0             jsb_r0r1        = JSB (REGISTER=0, REGISTER=1)
;  1045  0                                     : PRESERVE (1) NOTUSED (2,3,4,5,6,7,8,9,10,11);
```

```
;    1046  0      !
;    1047  0      ! Declare some common data structure initialization macros
;    1048  0      !
;    1049  0      MACRO
;    1050  0              ! Define shorthand for a single initialized dynamic string desc
;    1051  0              !
;    1052  0              $dyn_str_desc                           ! Static declaration
;  M 1053  0                      =
;  M 1054  0                      BLOCK [dsc$k_d_bln,BYTE]
;  M 1055  0                      PRESET ([dsc$b_class]   = dsc$k_class_d,
;  M 1056  0                              [dsc$b_dtype]   = dsc$k_dtype_t,
;  M 1057  0                              [dsc$w_length]  = 0,
;  M 1058  0                              [dsc$a_pointer] = 0 )
;    1059  0                      %,
;    1060  0
;    1061  0              $dyn_str_desc_init (desci)              ! Run-time initialization
;  M 1062  0                      =
;  M 1063  0                      BEGIN
;  M 1064  0                      BIND
;  M 1065  0                          desc = (desci) : VECTOR [2, LONG],
;  M 1066  0                          tmpl = exch$gq_dyn_str_template : VECTOR [2, LONG];
;  M 1067  0                      desc [0] = .tmpl [0];
;  M 1068  0                      desc [1] = .tmpl [1];
;  M 1069  0                      END
;    1070  0                      %,
;    1071  0
;    1072  0              ! Define macro for a single initialized static string desc.
;    1073  0              !
;    1074  0              $stat_str_desc (L, A)                   ! Static declaration, init to length and address
;  M 1075  0                      =
;  M 1076  0                      BLOCK [dsc$k_s_bln,BYTE]
;  M 1077  0                      PRESET( [dsc$b_class]   = dsc$k_class_s,
;  M 1078  0                              [dsc$b_dtype]   = dsc$k_dtype_t,
;  M 1079  0                              [dsc$w_length]  = (L),
;  M 1080  0                              [dsc$a_pointer] = (A) )
;    1081  0                      %,
;    1082  0
;    1083  0              $string_desc (str)                      ! Static declaration, init to length and address
;  M 1084  0                      =
;  M 1085  0                      BLOCK [dsc$k_s_bln,BYTE]
;  M 1086  0                      PRESET( [dsc$b_class]   = dsc$k_class_s,
;  M 1087  0                              [dsc$b_dtype]   = dsc$k_dtype_t,
;  M 1088  0                              [dsc$w_length]  = (%CHARCOUNT (STR)),
;  M 1089  0                              [dsc$a_pointer] = (UPLIT BYTE (STR)) )
;    1090  0                      %,
;    1091  0
;    1092  0              $stat_str_desc_init (desci, L, A)       ! Run-time initialization
;  M 1093  0                      =
;  M 1094  0                      BEGIN
;  M 1095  0                      BIND
;  M 1096  0                          desc = (desci) : BLOCK [, BYTE];
;  M 1097  0                      desc [dsc$b_class]   = dsc$k_class_s;
;  M 1098  0                      desc [dsc$b_dtype]   = dsc$k_dtype_t;
;  M 1099  0                      desc [dsc$w_length]  = (L);
;  M 1100  0                      desc [dsc$a_pointer] = (A);
;  M 1101  0                      END
;    1102  0                      %,
```

```
;    1103  0
;    1104  0          $str_desc_set (desci, L, A)     ! Copy new length and pointer fields (both static and dynamic)
;  M 1105  0                        =
;  M 1106  0                        BEGIN
;  M 1107  0                        BIND
;  M 1108  0                            desc = (desci) : BLOCK [, BYTE];
;  M 1109  0                        desc [dsc$w_length]  = (L);
;  M 1110  0                        desc [dsc$a_pointer] = (A);
;  M 1111  0                        END
;    1112  0                        %,
;    1113  0
;    1114  0          ! And shorthand for just a descriptor declaration
;    1115  0          !
;    1116  0          $desc_block
;  M 1117  0                        =
;  M 1118  0                        BLOCK [dsc$k_s_bln, BYTE]
;    1119  0                        %,
;    1120  0
;    1121  0          ! Short form for byte vector reference
;    1122  0          !
;    1123  0          $ref_bvector
;  M 1124  0                        =
;  M 1125  0                        REF $bvector
;    1126  0                        %,
;    1127  0
;    1128  0          ! Short form for byte block reference
;    1129  0          !
;    1130  0          $ref_bblock
;  M 1131  0                        =
;  M 1132  0                        REF $bblock
;    1133  0                        %;
;    1134  0
;    1135  0          STRUCTURE
;    1136  0                        $bvector [I; N] =
;    1137  0                                        [N]
;    1138  0                                        ($bvector+I)<0,8,0>;
;    1139  0
;    1140  0          !+
;    1141  0          ! SIGNAL_STOP a condition assuming no return.  LIB$STOP is not
;    1142  0          ! supposed to return, but BLISS doesn't krow this, so we block further
;    1143  0          ! flow here.  This will generate better code for us.
;    1144  0          !-
;    1145  0          MACRO
;    1146  0              $signal_stop []
;  M 1147  0                        =
;  M 1148  0                        BEGIN
;  M 1149  0                        LINKAGE
;  M 1150  0                            LNK = CALL : PRESERVE (0,1,2,3,4,5,6,7,8,9,10,11);
;  M 1151  0                        EXTERNAL ROUTINE
;  M 1152  0                            LIB$STOP : ADDRESSING_MODE (GENERAL) LNK NOVALUE;
;  M 1153  0                        BUILTIN
;  M 1154  0                            R0;
;  M 1155  0
;  M 1156  0                        LIB$STOP (%REMAINING);
;  M 1157  0                        RETURN (.R0);
;  M 1158  0
;  M 1159  0                        END
```

```
;    1160  0                          %;
;    1161  0
;    1162  0      !+
;    1163  0      ! SIGNAL a condition and return.
;    1164  0      !-
;    1165  0      MACRO
;    1166  0          $signal_return (code)
;M   1167  0                          =
;M   1168  0                          BEGIN
;M   1169  0                          LOCAL
;M   1170  0                              temp;
;M   1171  0
;M   1172  0                          temp = (code);          ! Need to avoid multiple calls, etc
;M   1173  0                          SIGNAL (.temp           ! Perform the actual signal of the error
;M   1174  0                                  %IF %LENGTH GTR 1 %THEN ,%REMAINING %FI);
;M   1175  0                          RETURN .temp
;M   1176  0
;M   1177  0                          END
;    1178  0                          %;
;    1179  0
;    1180  0      !+
;    1181  0      ! SIGNAL a condition and continue.
;    1182  0      !-
;    1183  0      MACRO
;    1184  0          $signal (code)
;M   1185  0                          =
;M   1186  0                          SIGNAL ( (code)         ! Perform the actual signal of the error
;M   1187  0                                 %IF %LENGTH GTR 1 %THEN ,%REMAINING %FI)
;    1188  0                          %;
;    1189  0      !+
;    1190  0      ! Check for a logic error.  If the expression is not true, then we have a problem.
;    1191  0      !-
;    1192  0      MACRO
;    1193  0          $logic_check (level, condition, error_code)
;M   1194  0                              =
;M   1195  0
;M   1196  0                              ! See if a compile time check is possible
;M   1197  0                              !
;M   1198  0                              %IF %CTCE ((condition))
;M   1199  0                              %THEN
;M   1200  0
;M   1201  0                                  ! The condition is a compile-time expression.  There is one special case, when the
;M   1202  0                                  ! condition is the string "(false)".  This is used as an unconditional logic abort.
;M   1203  0                                  ! If we have "(false)", then do a naked SIGNAL_STOP
;M   1204  0                                  !
;M   1205  0                                  %IF %IDENTICAL (condition, (false))
;M   1206  0                                  %THEN
;M   1207  0                                      SIGNAL_STOP (exch$_badlogic, 1, (error_code))
;M   1208  0
;M   1209  0                                  ! The condition is a normal test.  If it is true, print a message that the condition
;M   1210  0                                  ! was verified during compilation.  If false, generate a serious error.
;M   1211  0                                  !
;M   1212  0                                  %ELSE
;M   1213  0                                      %IF (condition)
;M   1214  0                                      %THEN
;M   1215  0                                          %PRINT ('assumption ',error_code,' verified during compilation')
;M   1216  0                                      %ELSE
```

```
; M 1217  0                                   %ERROR ('assumption ',error_code,' is not true')
; M 1218  0                               %FI
; M 1219  0                           %FI
; M 1220  0
; M 1221  0                           ! The condition is not a compile-time constant.  If the current variant calls for it,
; M 1222  0                           ! generate run-time code to test the assumption.
; M 1223  0                           !
; M 1224  0                           %ELSE
; M 1225  0                               %IF switch_variant GEQ (level)
; M 1226  0                               %THEN
; M 1227  0                                       BEGIN
; M 1228  0                                       IF NOT (condition)
; M 1229  0                                       THEN
; M 1230  0                                           SIGNAL_STOP (exch$_badlogic, 1, (error_code));
; M 1231  0                                       END
; M 1232  0                               %FI
; M 1233  0                           %FI
    1234  0                           %;
    1235  0
    1236  0      ! Message print routines
    1237  0      !
    1238  0      MACRO
    1239  0          $print_lit (string)
; M 1240  0                           =
; M 1241  0                           lib$put_output (%ASCID string)
    1242  0                           %,
    1243  0
    1244  0          $print_desc (desc)
; M 1245  0                           =
; M 1246  0                           lib$put_output (desc)
    1247  0                           %,
    1248  0
    1249  0          $print_fao (string)
; M 1250  0                           =
; M 1251  0                           BEGIN
; M 1252  0                           EXTERNAL ROUTINE  SHARE_FAO_BUFFER;
; M 1253  0                           lib$put_output (
; M 1254  0                               SHARE_FAO_BUFFER (%ASCID string
; M 1255  0                                   %IF %LENGTH GTR 1 %THEN ,%REMAINING %FI))
; M 1256  0                           END
    1257  0                           %;
    1258  0
    1259  0      ! Macros to manipulate queues
    1260  0      !
    1261  0      MACRO
    1262  0
    1263  0          ! Initialize the header of a queue.  This means make each of the 2 pointers in the header point to the header.
    1264  0          !
    1265  0          $queue_initialize (q_header)
; M 1266  0                           =
; M 1267  0                           BEGIN
; M 1268  0
; M 1269  0                           BIND
; M 1270  0                               _qh_ = (q_header) : VECTOR [2, LONG];
; M 1271  0
; M 1272  0                           _qh_ [0] = _qh_;
; M 1273  0                           _qh_ [1] = _qh_;
```

```
;  M 1274  0
;  M 1275  0                        END
;    1276  0                        %,
;    1277  0
;    1278  0        ! Insert an element at the head of a queue.
;    1279  0        !
;    1280  0        $queue_insert_head (item, q_header)
;  M 1281  0                        =
;  M 1282  0                        BEGIN
;  M 1283  0
;  M 1284  0                        BUILTIN
;  M 1285  0                            INSQUE;
;  M 1286  0                        BIND
;  M 1287  0                            _qh_ = (q_header) : VECTOR [2, LONG];
;  M 1288  0
;  M 1289  0                        INSQUE ((item), _qh_ [0])
;  M 1290  0
;  M 1291  0                        END
;    1292  0                        %,
;    1293  0
;    1294  0        ! Insert an element at the tail of a queue.
;    1295  0        !
;    1296  0        $queue_insert_tail ( tem, q_header)
;  M 1297  0                        =
;  M 1298  0                        BEGIN
;  M 1299  0
;  M 1300  0                        BUILTIN
;  M 1301  0                            INSQUE;
;  M 1302  0                        BIND
;  M 1303  0                            _qh_ = (q_header) : VECTOR [2, LONG];
;  M 1304  0
;  M 1305  0                        INSQUE ((item), ._qh_ [1])
;  M 1306  0
;  M 1307  0                        END
;    1308  0                        %,
;    1309  0        !+
;    1310  0        ! Remove the indicated element from a queue.  The first parameter is the address of the element.  The second
;    1311  0        ! parameter is optional.
;    1312  0        !
;    1313  0        ! If supplied, it is the address of a longword in which to store the element removed from the queue or 0 if
;    1314  0        ! no element was present in the queue.  The value of the expression is TRUE is a element was removed from the
;    1315  0        ! queue and FALSE otherwise.
;    1316  0        !
;    1317  0        ! If the second parameter is not supplied, the value of the expression is the address of the element removed
;    1318  0        ! from the queue or 0 if no element was present in the queue.
;    1319  0        !-
;    1320  0        $queue_remove (q_element, element)
;  M 1321  0                        =
;  M 1322  0                        BEGIN
;  M 1323  0
;  M 1324  0                        BIND
;  M 1325  0                            qhead_ = (q_element) : VECTOR [2, LONG];
;  M 1326  0                        BUILTIN
;  M 1327  0                            REMQUE;
;  M 1328  0
;  M 1329  0                        %IF (%NULL (element))
;  M 1330  0                        %THEN
```

```
;  M 1331  0                        LOCAL
;  M 1332  0                            _T_ : REF VECTOR [2, LONG];
;  M 1333  0                    %ELSE
;  M 1334  0                        BIND
;  M 1335  0                            _T_ = (element) : REF VECTOR [2, LONG];
;  M 1336  0                    %FI
;  M 1337  0
;  M 1338  0                    IF (REMQUE (_qhead_, _T_))
;  M 1339  0                    THEN
;  M 1340  0                        BEGIN
;  M 1341  0
;  M 1342  0                        ! queue was empty
;  M 1343  0                        !
;  M 1344  0                        IF (%NULL (element))
;  M 1345  0                        THEN
;  M 1346  0                            0
;  M 1347  0                        ELSE
;  M 1348  0                            (_T_ = 0; FALSE)
;  M 1349  0                        END
;  M 1350  0                    ELSE
;  M 1351  0                        BEGIN
;  M 1352  0
;  M 1353  0                        IF (%NULL (element))
;  M 1354  0                        THEN
;  M 1355  0                            ._T_
;  M 1356  0                        ELSE
;  M 1357  0                            true
;  M 1358  0                        END
;  M 1359  0                    END
;    1360  0                    %,
;    1361  0            !+
;    1362  0            ! Remove an element from the head of a queue.  The first parameter is the address of the queue header.  The
;    1363  0            ! second parameter is optional.
;    1364  0            !
;    1365  0            ! If supplied, it is the address of a longword in which to store the element removed from the queue or 0 if
;    1366  0            ! no element was present in the queue.  The value of the expression is TRUE is a element was removed from the
;    1367  0            ! queue and FALSE otherwise.
;    1368  0            !
;    1369  0            ! If the second parameter is not supplied, the value of the expression is the address of the element removed
;    1370  0            ! from the queue or 0 if no element was present in the queue.
;    1371  0            !-
;    1372  0            $queue_remove_head (q_header, element)
;  M 1373  0                               =
;  M 1374  0                               BEGIN
;  M 1375  0
;  M 1376  0                               BIND
;  M 1377  0                                   _qh_ = (q_header) : VECTOR [2, LONG];
;  M 1378  0
;  M 1379  0                               %IF (%NULL (element))
;  M 1380  0                               %THEN
;  M 1381  0                                   $queue_remove (._qh_ [0])
;  M 1382  0                               %ELSE
;  M 1383  0                                   $queue_remove (._qh_ [0], element)
;  M 1384  0                               %FI
;  M 1385  0
;  M 1386  0                               END
;    1387  0                               %,
```

```
;    1388  0          !+
;    1389  0          ! Remove an element from the tail of a queue.  The first parameter is the address of the queue header.  The
;    1390  0          ! second parameter is optional.
;    1391  0          !
;    1392  0          ! If supplied, it is the address of a longword in which to store the element removed from the queue or 0 if
;    1393  0          ! no element was present in the queue.  The value of the expression is TRUE is a element was removed from the
;    1394  0          ! queue and FALSE otherwise.
;    1395  0          !
;    1396  0          ! If the second parameter is not supplied, the value of the expression is the address of the element removed
;    1397  0          ! from the queue or 0 if no element was present in the queue.
;    1398  0          !-
;    1399  0          $queue_remove_tail (q_header, element)
;M   1400  0                              =
;M   1401  0                              BEGIN
;M   1402  0
;M   1403  0                              BIND
;M   1404  0                                  _qh_ = (q_header) : VECTOR [2, LONG];
;M   1405  0
;M   1406  0                              %IF (%NULL (element))
;M   1407  0                              %THEN
;M   1408  0                                  $queue_remove (._qh_ [1])
;M   1409  0                              %ELSE
;M   1410  0                                  $queue_remove (._qh_ [1], element)
;M   1411  0                              %FI
;M   1412  0                                                                                                              4F
;M   1413  0                              END
;    1414  0                              %,
;    1415  0
;    1416  0          ! Test a queue for emptiness.  TRUE if the queue is empty, FALSE if the queue is not empty            64
;    1417  0          !
;    1418  0          $queue_empty (q_header)
;M   1419  0                              =
;M   1420  0                              BEGIN
;M   1421  0
;M   1422  0                              BIND
;M   1423  0                                  _qh_ = (q_header) : VECTOR [2, LONG];
;M   1424  0
;M   1425  0                              _qh_ EQLA ._qh_ [0]
;M   1426  0
;M   1427  0                              END
;    1428  0                              %;
```

```
;  1429  0     !
;  1430  0     ! Define macros specific to OPCOM functions
;  1431  0     !
;  1432  0     MACRO
;  1433  0             !
;  1434  0             ! This next macro will allow us to examine any element
;  1435  0             ! of the count vector by just specifying its index.
;  1436  0             !
;  1437  0             OCD_W_ENABLECOUNT (N)    = (N*2)+$BYTEOFFSET(OCD_T_COUNTVECTOR),0,16,1%;
;  1438  0
;  1439  0     MACRO
;  1440  0             !++
;  1441  0             ! SCB_DEF
;  1442  0             !
;  1443  0             ! Functional description:
;  1444  0             !
;  1445  0             !       This macro will build the SCB table and the associated
;  1446  0             !       SCB's.  Each entry in the SCB table is a pointer to
;  1447  0             !       an SCB.  Each SCB describes a particular data structure.
;  1448  0             !       The table has a 1 origin, and is indexed by the data
;  1449  0             !       structure type.
;  1450  0             !
;  1451  0             ! Inputs:
;  1452  0             !
;  1453  0             !       DS_TYPE         : Data structure type.  An alphabetic
;  1454  0             !                         string that is assumed to prefix all
;  1455  0             !                         structure definitions and literals
;  1456  0             !                         pertaining to that structure type.
;  1457  0             !
;  1458  0             !       COUNT           : Count of data structures of this type
;  1459  0             !                         to be preallocated and inserted on
;  1460  0             !                         the its SCB look-aside list.
;  1461  0             !
;  1462  0             ! Implicit Input:
;  1463  0             !
;  1464  0             !       A COMPILETIME symbol, DS_TYPE_CODE, has
;  1465  0             !       been declared and initialized to 1.
;  1466  0             !
;  1467  0             ! Implicit Output:
;  1468  0             !
;  1469  0             !       A GLOBAL LITERAL defining the data
;  1470  0             !       structure type is declared.
;  1471  0             !--
;  1472  0             !
;  1473  0             SCB_DEF
;M 1474  0                     (DS_TYPE,COUNT) =
;M 1475  0                     GLOBAL LITERAL %NAME (DS_TYPE,'_K_TYPE') = COUNTER;
;M 1476  0                     PSECT OWN = $SCB_ENTRY;
;M 1477  0                     OWN
;M 1478  0                             SCB:    $bblock [SCB_K_SIZE]
;M 1479  0                                     PRESET  ([SCB_B_SIZE]           = %NAME (DS_TYPE,'_K_SIZE'),
;M 1480  0                                              [SCB_W_LAL_COUNT]      = COUNT,
;M 1481  0                                              [SCB_L_FLINK]          = SCB + $BYTEOFFSET (SCB_L_FLINK),
;M 1482  0                                              [SCB_L_BLINK]          = SCB + $BYTEOFFSET (SCB_L_FLINK)
;M 1483  0                                             );
;M 1484  0                     PSECT OWN = $SCB_TABLE;
;M 1485  0                     OWN
```

```
;  M 1486  0                    SCB_TBL:LONG INITIAL (SCB);
;  M 1487  0           UNDECLARE SCB, SCB_TBL;
;  M 1488  0           %ASSIGN (COUNTER,COUNTER+1);
;    1489  0           %;
```

```
;    1490  0      !
;    1491  0      ! Run-time library and other routines external to the facility
;    1492  0      !
;    1493  0      EXTERNAL ROUTINE
;    1494  0          cli$get_value       : ADDRESSING_MODE (GENERAL),              ! CLI Entity value fetch
;    1495  0          cli$present         : ADDRESSING_MODE (GENERAL),              ! CLI Entity presence boolean
;    1496  0          exe$alloc_csd       : ALLOC_CSD_ADDRESSING_MODE (GENERAL),    ! Allocate a CSD structure
;    1497  0          exe$chkrdacces      : ADDRESSING_MODE (GENERAL),              ! Check read access to pages
;    1498  0          exe$chkwrtacces     : ADDRESSING_MODE (GENERAL),              ! Check write acess
;    1499  0          exe$csp_call        : CSP_CALL_ADDRESSING_MODE (GENERAL),     ! Communicate with CSP
;    1500  0          exe$dealloc_csd     : DALLOC_CSD_ADDRESSING_MODE (GENERAL),   ! Release CSD structure
;    1501  0          exe$setopr          : ADDRESSING_MODE (GENERAL),              ! Set or clear OPR bit
;    1502  0          lib$free_vm         : ADDRESSING_MODE (GENERAL),              ! Release a block of virtual memory
;    1503  0          lib$get_vm          : ADDRESSING_MODE (GENERAL),              ! Allocate a block of virtual memory
;    1504  0          lib$lookup_key      : ADDRESSING_MODE (GENERAL),              ! Look up keyword in table
;    1505  0          lib$put_output      : ADDRESSING_MODE (GENERAL),              ! Display a line on SYS$OUTPUT
;    1506  0          lib$sig_to_ret      : ADDRESSING_MODE (GENERAL),              ! Convert a signal to a return
;    1507  0          ots$cvt_ti_l        : ADDRESSING_MODE (GENERAL),              ! Convert string (decimal) to integer
;    1508  0          str$copy_dx         : ADDRESSING_MODE (GENERAL),              ! Copy string of any class
;    1509  0          str$free1_dx        : ADDRESSING_MODE (GENERAL)               ! Release dynamic string
;    1510  0          ;
;    1511  0
;    1512  0      !
;    1513  0      ! Symbols from external modules which need explicit declarations
;    1514  0      !
;    1515  0      EXTERNAL LITERAL
;    1516  0          cli$_comma,                                                   ! Parameter ended with a comma
;    1517  0          cli$_concat,                                                  ! Parameter ended with a plus sign
;    1518  0          cli$_ivverb,                                                  ! Invalid verb
;    1519  0          cli$_locneg,                                                  ! An explicit /NOqual for local qual
;    1520  0          cli$_locpres,                                                 ! An explicit /qual for local qual
;    1521  0          cli$_negated,                                                 ! An explicit /NOqual was given
;    1522  0          cli$_nocomd,                                                  ! CLI saw a blank line and burped
;    1523  0          cli$_present,                                                 ! An explicit /qual was given
;    1524  0          cli$_facility;                                                ! CLI facility code
;    1525  0
;    1526  0      !
;    1527  0      ! External declarations for frequently referenced facility routines
;    1528  0      !
;    1529  0      EXTERNAL ROUTINE
;    1530  0          opc$free_vm,                                                  ! Free a chunk of virtual memory
;    1531  0          opc$get_vm;                                                   ! Allocate memory
;    1532  0      !
;    1533  0      ! Miscellaneous externals
;    1534  0      !
;    1535  0      EXTERNAL
;    1536  0          ascid_INVALIDRQCB : block [, BYTE];                           ! In a GLOBAL BIND in OPCOMDATA
;
;
;                                Library Statistics
;
;                                          -------- Symbols --------      Pages       Processing
;         File                             Total    Loaded   Percent      Mapped      Time
;
;    _$255$DUA28:[SYSLIB]LIB.L32;1          18619        32         0      1000          00:01.8
```

```
;                                       COMMAND QUALIFIERS

;       BLISS/LIBRARY=LIB$:/LIST=LIS$: SRC$:OPCOMLIB

; Run Time:         00:13.2
; Elapsed Time:     01:03.5
; Lines/CPU Min:     6971
; Lexemes/CPU-Min: 37161
; Memory Used:  168 pages
; Library Precompilation Complete
```

LOGFILE
LIS

OPCOMDEF
LIS

OPCOMDATA
LIS

OPCOMINI
LIS

OPCOMUTIL
LIS

OPCOMLIB
LIS

OPCDEFTMP
LIS

OPCOMMAIN
LIS

OPCOMOLD
LIS

OPCOMRPLY
LIS

OPCCRASH
LIS

OPERUTIL
LIS

OPCOMROST
LIS