

[illegible]

```
000000  PPPPPPP  CCCCCCCC  000000  MM  MM  LL  IIIII  88888888
000000  PPPPPPP  CCCCCCCC  000000  MM  MM  LL  IIIII  88888888
00  00  PP  PP  CC  00  00  MMMM  MMMM  LL  II  88  88
00  00  PP  PP  CC  00  00  MMMM  MMMM  LL  II  88  88
00  00  PP  PP  CC  00  00  MM  MM  LL  II  88  88
00  00  PP  PP  CC  00  00  MM  MM  LL  II  88  88
00  00  PPPPPPP  CC  00  00  MM  MM  LL  II  88888888
00  00  PPPPPPP  CC  00  00  MM  MM  LL  II  88888888
00  00  PP  CC  00  00  MM  MM  LL  II  88  88
00  00  PP  CC  00  00  MM  MM  LL  II  88  88
00  00  PP  CC  00  00  MM  MM  LL  II  88  88
00  00  PP  CC  00  00  MM  MM  LL  II  88  88
000000  PP  CCCCCCCC  000000  MM  MM  LLLLLLLLLL  IIIII  88888888
000000  PP  CCCCCCCC  000000  MM  MM  LLLLLLLLLL  IIIII  88888888
                                ....
                                ....
                                ....
                                ....
```

```
88888888  333333  222222
88888888  333333  222222
88  88  33  33  22  22
88  88  33  33  22  22
88  88  33  33  22  22
88888888  33  22
88888888  33  22
88  88  33  22
88  88  33  22
88  88  33  22
88888888  333333  2222222222
88888888  333333  2222222222
```


MODULE OPC\$OPCOMLIB (IDENT = 'V04-000') = %TITLE 'Facility-wide library module'
BEGIN

```
*****
*
*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
*  ALL RIGHTS RESERVED.
*
*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
*  ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
*  OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
*  TRANSFERRED.
*
*  THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
*  AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
*  CORPORATION.
*
*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
*
*****
```

```
--
FACILITY:      OPCOM - Operator communication facility
ABSTRACT:      BLISS Library for OPCOM facility
ENVIRONMENT:   VAX/VMS User mode
AUTHOR:        CW Hobbs
CREATED:       24-Aug-1983
MODIFIED BY:
    V03-003 CWH3003      CW Hobbs      12-Aug-1984
             Remove REPLY and SOFTWARE operators from the known list
    V03-002 CWH3002      CW Hobbs      16-Sep-1983
             Move non-facility EXTERNAL ROUTINE declarations to this file
--
```

Include the data structure definitions from external files

```
LIBRARY 'SYS$LIBRARY:LIB';
REQUIRE 'LIB$OPCDEFTMP';      ! New message definitions
REQUIRE 'LIB$OPCOMDEF';
REQUIRE 'SHRLIB$CLUMBX';
```

```
! Define a compile time variable to be used as a table origin by several of the macros.
```

```
COMPILETIME      counter = 0;
```

```
! Define literal values
```

```
LITERAL
```

```
max_dev_nam      = 64,                ! Maximum length of a dev name
```

```
! Define volume protection masks that are used to control access to the operator mailbox.
```

```
read_write       = %X'OFFOF',         ! Allow read and write access
read_nowrite     = %X'OFFFF',         ! Allow read, don't allow write
```

```
! Other useful literals
```

```
true            = 1,                  ! BOOLEAN value
false           = 0,                  ! BOOLEAN value
```

```
on              = 1,                  ! Bit value
off             = 0,                  ! Bit value
```

```
! Define the masks containing all known operators
```

```
known_attn_mask1 = (OPCSM_NM_CENTRL OR  OPCS_M_NM_PRINT OR  OPCS_M_NM_TAPES OR
                    OPCS_M_NM_DISKS OR  OPCS_M_NM_DEVICE OR  OPCS_M_NM_CARDS OR
                    OPCS_M_NM_NETWORK OR OPCS_M_NM_CLUSTER OR OPCS_M_NM_SECURITY OR
                    !OPCS_M_NM_REPLY OR  OPCS_M_NM_SOFTWARE OR OPCS_M_NM_FILL_11 OR
                    OPCS_M_NM_OPER1 OR  OPCS_M_NM_OPER2 OR  OPCS_M_NM_OPER3 OR
                    OPCS_M_NM_OPER4 OR  OPCS_M_NM_OPER5 OR  OPCS_M_NM_OPER6 OR
                    OPCS_M_NM_OPER7 OR  OPCS_M_NM_OPER8 OR  OPCS_M_NM_OPER9 OR
                    OPCS_M_NM_OPER10 OR OPCS_M_NM_OPER11 OR OPCS_M_NM_OPER12),
```

```
known_attn_mask2 = 0;
```

```
! Define routine linkages
```

```
LINKAGE
```

```
alloc_csd        = JSB (REGISTER=1; REGISTER=2)
                  : NOPRESERVE (3) NOTUSED (4,5,6,7,8,9,10,11),
csp_call         = JSB (REGISTER=2)
                  : NOTUSED (3,4,5,6,7,8,9,10,11),
dalloc_csd       = JSB (REGISTER=0)
                  : NOPRESERVE (2,3) NOTUSED (4,5,6,7,8,9,10,11),
jsb_r0r1         = JSB (REGISTER=0, REGISTER=1)
                  : PRESERVE (1) NOTUSED (2,3,4,5,6,7,8,9,10,11);
```


! Declare some common data structure initialization macros

MACRO

! Define shorthand for a single initialized dynamic string desc

```
$dyn_str_desc                                ! Static declaration
=
  BLOCK [dsc$k_d_bln,BYTE]
  PRESET ([dsc$b_class] = dsc$k_class_d,
          [dsc$b_dtype] = dsc$k_dtype_t,
          [dsc$w_length] = 0,
          [dsc$a_pointer] = 0)
%,

$dyn_str_desc_init (desci)                  ! Run-time initialization
=
  BEGIN
  BIND
    desc = (desci) : VECTOR [2, LONG],
    tmp1 = exch$gg_dyn_str_template : VECTOR [2, LONG];
    desc [0] = .tmp1 [0];
    desc [1] = .tmp1 [1];
  END
%,
```

! Define macro for a single initialized static string desc.

```
$stat_str_desc (L, A)                      ! Static declaration, init to length and address
=
  BLOCK [dsc$k_s_bln,BYTE]
  PRESET( [dsc$b_class] = dsc$k_class_s,
          [dsc$b_dtype] = dsc$k_dtype_t,
          [dsc$w_length] = (L),
          [dsc$a_pointer] = (A) )
%,

$string_desc (str)                        ! Static declaration, init to length and address
=
  BLOCK [dsc$k_s_bln,BYTE]
  PRESET( [dsc$b_class] = dsc$k_class_s,
          [dsc$b_dtype] = dsc$k_dtype_t,
          [dsc$w_length] = (%CHARCOUNT (STR)),
          [dsc$a_pointer] = (UPLIT BYTE (STR)) )
%,
```

```
$stat_str_desc_init (desci, L, A)          ! Run-time initialization
=
  BEGIN
  BIND
    desc = (desci) : BLOCK [, BYTE];
    desc [dsc$b_class] = dsc$k_class_s;
    desc [dsc$b_dtype] = dsc$k_dtype_t;
    desc [dsc$w_length] = (L);
    desc [dsc$a_pointer] = (A);
  END
```

```

%,
$str_desc_set (desci, L, A)      ! Copy new length and pointer fields (both static and dynamic)
=
BEGIN
BIND
    desc = (desci) : BLOCK [, BYTE];
    desc [dsc$w_length] = (L);
    desc [dsc$a_pointer] = (A);
END
%,

! And shorthand for just a descriptor declaration
! $desc_block
=
BLOCK [dsc$k_s_bln, BYTE]
%,

! Short form for byte vector reference
! $ref_bvector
=
REF $bvector
%,

! Short form for byte block reference
! $ref_bblock
=
REF $bblock
%,

STRUCTURE
    $bvector [I; N] =
        [N]
        ($bvector+1)<0,8,0>;

```

```

!+
! SIGNAL_STOP a condition assuming no return. LIB$STOP is not
! supposed to return, but BLISS doesn't know this, so we block further
! flow here. This will generate better code for us.
!-

```

```

MACRO
    $signal_stop []
        =
        BEGIN
        LINKAGE
            LNK = CALL : PRESERVE (0,1,2,3,4,5,6,7,8,9,10,11);
        EXTERNAL ROUTINE
            LIB$STOP : ADDRESSING_MODE (GENERAL) LNK NOVALUE;
        BUILTIN
            R0;

        LIB$STOP (%REMAINING);
        RETURN (.R0);
    
```


END
%;

!+
!- SIGNAL a condition and return.

```
MACRO
    $signal_return (code)
        =
        BEGIN
        LOCAL
            temp;

            temp = (code);           ! Need to avoid multiple calls, etc
            SIGNAL (.temp)           ! Perform the actual signal of the error
            %IF %LENGTH GTR 1 %THEN ,%REMAINING %FI);
            RETURN .temp

        END
        %;
```

!+
!- SIGNAL a condition and continue.

```
MACRO
    $signal (code)
        =
        SIGNAL ( (code)             ! Perform the actual signal of the error
            %IF %LENGTH GTR 1 %THEN ,%REMAINING %FI)
        %;
```

!+
!- Check for a logic error. If the expression is not true, then we have a problem.

```
MACRO
    $logic_check (level, condition, error_code)
        =
        ! See if a compile time check is possible
        !-
        %IF %CTCE ((condition))
        %THEN
            ! The condition is a compile-time expression. There is one special case, when the
            ! condition is the string "(false)". This is used as an unconditional logic abort.
            ! If we have "(false)", then do a naked SIGNAL_STOP
            %IF %IDENTICAL (condition, (false))
            %THEN
                SIGNAL_STOP (exch$_badlogic, 1, (error_code))

            ! The condition is a normal test. If it is true, print a message that the condition
            ! was verified during compilation. If false, generate a serious error.
            !-
            %ELSE
                %IF (condition)
```

```

%THEN
%PRINT ('assumption ',error_code,' verified during compilation')
%ELSE
%ERROR ('assumption ',error_code,' is not true')
%FI

```

```

! The condition is not a compile-time constant. If the current variant calls for it,
! generate run-time code to test the assumption.

```

```

%ELSE
%IF switch_variant GEQ (level)
%THEN
BEGIN
IF NOT (condition)
THEN
SIGNAL_STOP (exch$_badlogic, 1, (error_code));
END
%FI
%FI
%:

```

```

! Message print routines

```

```

MACRO
$print_lit (string)
=
lib$put_output (%ASCID string)
%,

$print_desc (desc)
=
lib$put_output (desc)
%,

$print_fao (string)
=
BEGIN
EXTERNAL ROUTINE SHARE_FAO_BUFFER;
lib$put_output (
SHARE_FAO_BUFFER (%ASCID string
%IF %LENGTH GTR 1 %THEN ,%REMAINING %FI))
END
%:

```

```

! Macros to manipulate queues

```

```

MACRO
! Initialize the header of a queue. This means make each of the 2 pointers in the header point to the header.
!
$queue_initialize (q_header)
=
BEGIN
BIND

```



```

        _qh_ = (q_header) : VECTOR [2, LONG];
        _qh_ [0] = _qh_;
        _qh_ [1] = _qh_;
    END
    %,

```

! Insert an element at the head of a queue.

```

$queue_insert_head (item, q_header)
=
BEGIN
    BUILTIN
    INSQUE;
    BIND
        _qh_ = (q_header) : VECTOR [2, LONG];
    INSQUE ((item), _qh_ [0])
    END
    %,

```

! Insert an element at the tail of a queue.

```

$queue_insert_tail (item, q_header)
=
BEGIN
    BUILTIN
    INSQUE;
    BIND
        _qh_ = (q_header) : VECTOR [2, LONG];
    INSQUE ((item), _qh_ [1])
    END
    %,

```

+ Remove the indicated element from a queue. The first parameter is the address of the element. The second parameter is optional.

If supplied, it is the address of a longword in which to store the element removed from the queue or 0 if no element was present in the queue. The value of the expression is TRUE if an element was removed from the queue and FALSE otherwise.

If the second parameter is not supplied, the value of the expression is the address of the element removed from the queue or 0 if no element was present in the queue.

```

$queue_remove (q_element, element)
=
BEGIN
    BIND
        _qhead_ = (q_element) : VECTOR [2, LONG];

```

```

BUILTIN
  REMQUE;
%IF (%NULL (element))
%THEN
  LOCAL
    _T_ : REF VECTOR [2, LONG];
%ELSE
  BIND
    _T_ = (element) : REF VECTOR [2, LONG];
%FI
IF (REMQUE (_qhead_, _T_))
THEN
  BEGIN
    ! queue was empty
    IF (%NULL (element))
    THEN
      0
    ELSE
      (_T_ = 0; FALSE)
    END
  ELSE
  BEGIN
    IF (%NULL (element))
    THEN
      _T_
    ELSE
      true
    END
  END
END
%,

```

+ Remove an element from the head of a queue. The first parameter is the address of the queue header. The second parameter is optional.

If supplied, it is the address of a longword in which to store the element removed from the queue or 0 if no element was present in the queue. The value of the expression is TRUE if an element was removed from the queue and FALSE otherwise.

If the second parameter is not supplied, the value of the expression is the address of the element removed from the queue or 0 if no element was present in the queue.

```

_queue_remove_head (q_header, element)
=
BEGIN
  BIND
    _qh_ = (q_header) : VECTOR [2, LONG];
  %IF (%NULL (element))
  %THEN
    _queue_remove (._qh_ [0])

```



```

%ELSE
  $queue_remove (._qh_ [0], element)
%FI
END
%,

```

+ Remove an element from the tail of a queue. The first parameter is the address of the queue header. The second parameter is optional.

If supplied, it is the address of a longword in which to store the element removed from the queue or 0 if no element was present in the queue. The value of the expression is TRUE if an element was removed from the queue and FALSE otherwise.

If the second parameter is not supplied, the value of the expression is the address of the element removed from the queue or 0 if no element was present in the queue.

```

_$queue_remove_tail (q_header, element)
=
BEGIN
  BIND
    _qh_ = (q_header) : VECTOR [2, LONG];
  %IF (%NULL (element))
  %THEN
    $queue_remove (._qh_ [1])
  %ELSE
    $queue_remove (._qh_ [1], element)
  %FI
END
%,

```

! Test a queue for emptiness. TRUE if the queue is empty, FALSE if the queue is not empty

```

_$queue_empty (q_header)
=
BEGIN
  BIND
    _qh_ = (q_header) : VECTOR [2, LONG];
  _qh_ EQLA ._qh_ [0]
END
%:

```


Define macros specific to OPCOM functions

MACRO

This next macro will allow us to examine any element of the count vector by just specifying its index.

OCD_W_ENABLECOUNT (N) = (N*2)+\$BYTEOFFSET(OCD_T_COUNTVECTOR),0,16,1%;

MACRO

++
SCB_DEF

Functional description:

This macro will build the SCB table and the associated SCB's. Each entry in the SCB table is a pointer to an SCB. Each SCB describes a particular data structure. The table has a 1 origin, and is indexed by the data structure type.

Inputs:

DS_TYPE : Data structure type. An alphabetic string that is assumed to prefix all structure definitions and literals pertaining to that structure type.

COUNT : Count of data structures of this type to be preallocated and inserted on the its SCB look-aside list.

Implicit Input:

A COMPILETIME symbol, DS_TYPE_CODE, has been declared and initialized to 1.

Implicit Output:

A GLOBAL LITERAL defining the data structure type is declared.

--
SCB_DEF

```
(DS_TYPE,COUNT) =
GLOBAL LITERAL %NAME (DS_TYPE,'_K_TYPE') = COUNTER;
PSECT OWN = $SCB_ENTRY;
OWN      SCB:  $bblock [SCB_K_SIZE]
              PRESET ([SCB_Q_SIZE]          = %NAME (DS_TYPE,'_K_SIZE'),
                      [SCB_W_LAL COUNT]      = COUNT,
                      [SCB_L_FLINK]          = SCB + $BYTEOFFSET (SCB_L_FLINK),
                      [SCB_L_BLINK]          = SCB + $BYTEOFFSET (SCB_L_FLINK)
                      );
PSECT OWN = $SCB_TABLE;
```



```
OWN      SCB_TBL:LONG INITIAL (SCB);  
UNDECLARE SCB, SCB_TBL;  
%ASSIGN (COUNTER,COUNTER+1);  
%;
```


Run-time library and other routines external to the facility

EXTERNAL ROUTINE

cli\$get_value	: ADDRESSING_MODE (GENERAL),	: CLI Entity value fetch
cli\$present	: ADDRESSING_MODE (GENERAL),	: CLI Entity presence boolean
exe\$alloc_csd	: ALLOC_CSD ADDRESSING_MODE (GENERAL),	: Allocate a CSD structure
exe\$chkrdacces	: ADDRESSING_MODE (GENERAL),	: Check read access to pages
exe\$chkwrtacces	: ADDRESSING_MODE (GENERAL),	: Check write access
exe\$csp_call	: CSP_CALL ADDRESSING_MODE (GENERAL),	: Communicate with CSP
exe\$dealloc_csd	: DEALLOC_CSD ADDRESSING_MODE (GENERAL),	: Release CSD structure
exe\$setopr	: ADDRESSING_MODE (GENERAL),	: Set or clear OPR bit
lib\$free_vm	: ADDRESSING_MODE (GENERAL),	: Release a block of virtual memory
lib\$get_vm	: ADDRESSING_MODE (GENERAL),	: Allocate a block of virtual memory
lib\$lookup_key	: ADDRESSING_MODE (GENERAL),	: Look up keyword in table
lib\$put_output	: ADDRESSING_MODE (GENERAL),	: Display a line on SYSS\$OUTPUT
lib\$sig_to_ret	: ADDRESSING_MODE (GENERAL),	: Convert a signal to a return
ots\$cvt_ti_l	: ADDRESSING_MODE (GENERAL),	: Convert string (decimal) to integer
str\$copy_dx	: ADDRESSING_MODE (GENERAL),	: Copy string of any class
str\$freeT_dx	: ADDRESSING_MODE (GENERAL)	: Release dynamic string

Symbols from external modules which need explicit declarations

EXTERNAL LITERAL

cli\$_comma,	: Parameter ended with a comma
cli\$_concat,	: Parameter ended with a plus sign
cli\$_ivverb,	: Invalid verb
cli\$_locneg,	: An explicit /NOqual for local qual
cli\$_locpres,	: An explicit /qual for local qual
cli\$_negated,	: An explicit /NOqual was given
cli\$_nocomd,	: CLI saw a blank line and burped
cli\$_present,	: An explicit /qual was given
cli\$_facility;	: CLI facility code

External declarations for frequently referenced facility routines

EXTERNAL ROUTINE

opc\$free_vm,	: Free a chunk of virtual memory
opc\$get_vm;	: Allocate memory

Miscellaneous externals

EXTERNAL

ascid_INVALIDRQCB : block [, BYTE];	: In a GLOBAL BIND in OPCOMDATA
-------------------------------------	---------------------------------

0288 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

