


```

CCCCCCCC LL      UU      UU      SSSSSSSS CCCCCCCC 000000 MM      MM      MM      MM
CCCCCCCC LL      UU      UU      SSSSSSSS CCCCCCCC 000000 MM      MM      MM      MM
CC        LL      UU      UU      SS        CC        00      00      MMMM  MMMM  MMMM  MMMM
CC        LL      UU      UU      SS        CC        00      00      MMMM  MMMM  MMMM  MMMM
CC        LL      UU      UU      SS        CC        00      00      MM   MM   MM   MM   MM   MM
CC        LL      UU      UU      SS        CC        00      00      MM   MM   MM   MM   MM   MM
CC        LL      UU      UU      SSSSSS   CC        00      00      MM   MM   MM   MM   MM   MM
CC        LL      UU      UU      SSSSSS   CC        00      00      MM   MM   MM   MM   MM   MM
CC        LL      UU      UU      SS        CC        00      00      MM   MM   MM   MM   MM   MM
CC        LL      UU      UU      SS        CC        00      00      MM   MM   MM   MM   MM   MM
CC        LL      UU      UU      SS        CC        00      00      MM   MM   MM   MM   MM   MM
CC        LL      UU      UU      SS        CC        00      00      MM   MM   MM   MM   MM   MM
CCCCCCCC LL      UU      UU      SSSSSSSS CCCCCCCC 000000 MM   MM   MM   MM   MM   MM
CCCCCCCC LL      UU      UU      SSSSSSSS CCCCCCCC 000000 MM   MM   MM   MM   MM   MM

```

```

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
LLLLLLLL IIIIII SSSSSSSS
LLLLLLLL IIIIII SSSSSSSS

```

```

1 0001 0 MODULE OPC$CLUSCOMM (
2 0002 0 LANGUAGE (BLISS32),
3 0003 0 IDENT = 'V04-000'
4 0004 0 ) =
5 0005 0
6 0006 0 *****
7 0007 0 *
8 0008 0 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY *
9 0009 0 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS. *
10 0010 0 * ALL RIGHTS RESERVED. *
11 0011 0 *
12 0012 0 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED *
13 0013 0 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE *
14 0014 0 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER *
15 0015 0 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY *
16 0016 0 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY *
17 0017 0 * TRANSFERRED. *
18 0018 0 *
19 0019 0 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE *
20 0020 0 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT *
21 0021 0 * CORPORATION. *
22 0022 0 *
23 0023 0 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS *
24 0024 0 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL. *
25 0025 0 *
26 0026 0 *
27 0027 0 *****
28 0028 0
29 0029 0 ++
30 0030 0 FACILITY:
31 0031 0
32 0032 0 OPCOM
33 0033 0
34 0034 0 ABSTRACT:
35 0035 0
36 0036 0 This module contains communications routines used by cluster functions within OPCOM.
37 0037 0
38 0038 0 Environment:
39 0039 0
40 0040 0 VAX/VMS operating system.
41 0041 0
42 0042 0 Author:
43 0043 0
44 0044 0 CW Hobbs
45 0045 0
46 0046 0 Creation date:
47 0047 0
48 0048 0 14 July 1983
49 0049 0
50 0050 0 Revision history:
51 0051 0
52 0052 0 V03-004 CWH3004 CW Hobbs 18-May-1984
53 0053 0 Reduce csp messages to two total, one per node to avoid
54 0054 0 a temporary problem with port overloads.
55 0055 0
56 0056 0 V03-003 CWH3169 CW Hobbs 5-May-1984
57 0057 0 Second pass for cluster-wide OPCOM:

```



```

70      0069 1 BEGIN                               : Start of CLUSCOMM
71      0070 1
72      0071 1 LIBRARY 'SYS$LIBRARY:LIB.L32';
73      0072 1 LIBRARY 'LIB$:OPCOMLIB';
74      0073 1 REQUIRE 'SHRLIB$:CSPDEF';
75      0267 1
76      0268 1 FORWARD ROUTINE
77      0269 1 CLUSCOMM_COD_ALLOCATE,           : Allocate a cluster output descriptor
78      0270 1 CLUSCOMM_COD_ERROR : NOVALUE,     : Handle an error described by a cod
79      0271 1 CLUSCOMM_COD_ERROR_AST : NOVALUE,  : User mode ast routine for a cod error
80      0272 1 CLUSCOMM_DECLARE_KERNEL_AST,      : Declare kernel AST to start things moving
81      0273 1 CLUSCOMM_OUTPUT_KERNEL_AST,      : Handle ast from CSP
82      0274 1 CLUSCOMM_SEND,                   : Jacket routine to send message to remote node(s)
83      0275 1 CLUSCOMM_SEND ONE,               : Send message to single remote node
84      0276 1 CLUSCOMM_TARGET_IN_QUEUE;        : Count number of times target node in queue
85      0277 1
86      0278 1 EXTERNAL ROUTINE
87      0279 1 CLUSUTIL_FIND_NOD_BY_CSID,
88      0280 1 CLUSUTIL_NODE_MESSAGE,
89      0281 1 DUMP_LOG_FILE,
90      0282 1 SHARE_FAB_BUFFER,
91      0283 1 WRITE_LOG_FILE;
92      0284 1
93      0285 1 GLOBAL                               : Global so that SDA can find them easily
94      0286 1 COD_ALLOCATED,                     : Count of CODs created
95      0287 1 COD_BUSY_COUNT,                   : Current count of i/os pending
96      0288 1 COD_BUSY_MAX : INITIAL (2),       : Maximum number of EXEC$CSP_CALLs pending
97      0289 1 COD_BUSY_NODE : INITIAL (1),      : Maximum number of EXEC$CSP_CALLs pending to single node
98      0290 1 COD_ERRORS,                       : Count of requests with errors
99      0291 1 COD_FLUSHED,                     : Count of requests flushed (also count as errors)
100     0292 1 COD_REQUESTS,                    : Count of requests made
101     0293 1 COD_QUEUED,                      : Count of requests queued
102     0294 1 COD_BUSY_QUEUE : VECTOR [2, LONG] : Queue of CODs pending for I/O
103     0295 1 INITIAL (REP 2 OF (COD_BUSY_QUEUE)),
104     0296 1 COD_FREE_QUEUE : VECTOR [2, LONG] : Queue of cods available for use
105     0297 1 INITIAL (REP 2 OF (COD_FREE_QUEUE)),
106     0298 1 COD_WAIT_QUEUE : VECTOR [2, LONG] : Queue of cods waiting for actual EXEC$CSP_CALL to be queued
107     0299 1 INITIAL (REP 2 OF (COD_WAIT_QUEUE)),
108     0300 1 COD_GARBAGE_QUEUE : VECTOR [2, LONG] : Pointer to list of virtual memory to deallocate
109     0301 1 INITIAL (REP 2 OF (COD_GARBAGE_QUEUE));
110     0302 1
111     0303 1 ! A macro to put virtual memory back on the queue of garbage to be deallocated
112     0304 1
113     0305 1 MACRO
114     0306 1 COLLECT_GARBAGE (INP_DESC) =
115     0307 1 BEGIN
116     0308 1 BIND
117     0309 1 desc = (INP_DESC) : VECTOR [, LONG],
118     0310 1 garbage = .desc [1] : VECTOR [, LONG];
119     0311 1 garbage [2] = .desc [0]; ! Store length as second longword in block
120     0312 1 $queue_insert_tail (garbage, cod_garbage_queue);
121     0313 1 END %

```

```

123 0314 1 GLOBAL ROUTINE cluscomm_cod_allocate =
124 0315 1
125 0316 1 !**
126 0317 1 Functional descriptor:
127 0318 1
128 0319 1 This routine allocates a COD for a cluster write
129 0320 1
130 0321 1 Input:
131 0322 1 None.
132 0323 1
133 0324 1 Output:
134 0325 1 None.
135 0326 1
136 0327 1 Routine Value:
137 0328 1 Address of block allocated
138 0329 1 --
139 0330 1
140 0331 2 BEGIN ! Start of cluscomm_cod_allocate
141 0332 2
142 0333 2 LOCAL
143 0334 2 cod : $ref bblock, ! cod data structure
144 0335 2 garb : REF VECTOR [, LONG],
145 0336 2 ptr,
146 0337 2 status;
147 0338 2
148 0339 2
149 0340 2 ! If any garbage nodes are in the hopper, send them away. Garbage is reclaimed this
150 0341 2 way so that the kernel ast routines do not do free_vm calls on memory allocated
151 0342 2 from user mode.
152 0343 2
153 0344 2 $queue_remove_head (cod_garbage_queue, garb);
154 0345 2 WHILE .garb NEQ 0
155 0346 2 DO
156 0347 3 BEGIN
157 0348 4 IF NOT (status = opc$free_vm (garb [2], garb))
158 0349 3 THEN
159 0350 3 $signal_stop (.status);
160 0351 3 $queue_remove_head (cod_garbage_queue, garb);
161 0352 2 END;
162 0353 2
163 0354 2 ! Get a cod, a Cluster Output Descriptor, if none available on the queue then
164 0355 2 allocate and initialize one.
165 0356 2
166 0357 2 $queue_remove_head (cod_free_queue, cod);
167 0358 2 IF .cod EQL 0
168 0359 2 THEN
169 0360 3 BEGIN
170 0361 4 IF NOT (status = opc$get_vm (%ref (cod_k_size), ptr))
171 0362 3 THEN
172 0363 3 $signal_stop (.STATUS);
173 0364 3 cod_allocated = .cod_allocated + 1;
174 0365 3 cod = .ptr;
175 0366 3 CH$FILL (0, cod_k_size, .cod);
176 0367 3 cod [cod_w_size] = cod_k_size;
177 0368 3 cod [cod_b_type] = %x'77';
178 0369 2 END;
179 0370 2

```

```

: 180 0371 2 ; Init the block
: 181 0372 2 ;
: 182 0373 2 (cod [cod_q_quetime]) = 0;
: 183 0374 2 (cod [cod_q_quetime]+4) = 0;
: 184 0375 2 cod [cod_a_csd] = 0;
: 185 0376 2 cod [cod_l_msglen] = 0;
: 186 0377 2
: 187 0378 2 RETURN .cod;
: 188 0379 1 END;

```

. End of cluscomm_cod_allocate

```

.TITLE OPC$CLUSCOMM
.IDENT \V04-000\
.PSECT $GLOBALS,NOEXE,2

```

```

00000 COD_ALLOCATED::
      .BLKB 4
00004 COD_BUSY_COUNT::
      .BLKB 4
00000002 00008 COD_BUSY_MAX::
      .LONG 2
00000001 0000C COD_BUSY_NODE::
      .LONG 1
00010 COD_ERRORS::
      .BLKB 4
00014 COD_FLUSHED::
      .BLKB 4
00018 COD_REQUESTS::
      .BLKB 4
0001C COD_QUEUED::
      .BLKB 4
00000000' 00020 COD_BUSY_QUEUE::
      .ADDRESS COD_BUSY_QUEUE
00000000' 00024 .ADDRESS COD_BUSY_QUEUE
00000000' 00028 COD_FREE_QUEUE::
      .ADDRESS COD_FREE_QUEUE
00000000' 0002C .ADDRESS COD_FREE_QUEUE
00000000' 00030 COD_WAIT_QUEUE::
      .ADDRESS COD_WAIT_QUEUE
00000000' 00034 .ADDRESS COD_WAIT_QUEUE
00000000' 00038 COD_GARBAGE_QUEUE::
      .ADDRESS COD_GARBAGE_QUEUE
00000000' 0003C .ADDRESS COD_GARBAGE_QUEUE

-QH = COD_GARBAGE_QUEUE
-QH = COD_GARBAGE_QUEUE
-QH = COD_FREE_QUEUE
.EXTRN CLUSUTIL_FIND_NODE_BY_CSID
.EXTRN CLUSUTIL_NODE_MESSAGE
.EXTRN DUMP_LOG_FILE, SHARE_FAO_BUFFER
.EXTRN WRITE_LOG_FILE, OPC$FREE_VM
.EXTRN LIB$STOP, OPC$GET_VM

.PSECT $CODE$,NOWRT,2

007C 00000 .ENTRY CLUSCOMM_COD_ALLOCATE, Save R2,R3,R4,R5,R6 ; 0314

```



```

: 190      0380 1 GLOBAL ROUTINE cluscomm_declare_kernel_ast =
: 191      0381 1
: 192      0382 1 :++
: 193      0383 1 : Functional description:
: 194      0384 1 :
: 195      0385 1 :     This routine declares an ast to start the I/O, both it and the AST operate in kernel mode
: 196      0386 1 :
: 197      0387 1 : Input:
: 198      0388 1 :     None.
: 199      0389 1 :
: 200      0390 1 : Output:
: 201      0391 1 :     None.
: 202      0392 1 :
: 203      0393 1 : Routine Value:
: 204      0394 1 :     Value from DCLAST
: 205      0395 1 : --
: 206      0396 1
: 207      0397 2 BEGIN                               ! Start of cluscomm_declare_kernel_ast
: 208      0398 2
: 209      0399 2 RETURN $DCLAST (ASTADR=cluscomm_output_kernel_ast, ASTPRM=0); ! 0 means start
: 210      0400 2
: 211      0401 1 END;                               ! End of cluscomm_declare_kernel_ast

```

.EXTRN SYSSDCLAST

		0000 00000	.ENTRY CLUSCOMM_DECLARE_KERNEL_AST, Save nothing	: 0380
		7E 7C 00002	CLRQ -(SP)	: 0399
		CF 9F 00004	PUSHAB CLUSCOMM_OUTPUT_KERNEL_AST	:
00000000G 00	0000V	03 FB 00008	CALLS #3, SYSSDCLAST	:
		04 0000F	RET	: 0401

: Routine Size: 16 bytes, Routine Base: \$CODE\$ + 007A

```
0402 1 GLOBAL ROUTINE cluscomm_output_kernel_ast (csd : $ref_bblock) : NOVALUE =
0403 1
0404 1  !++
0405 1  Functional description:
0406 1
0407 1      This routine is the I/O completion for a EXE$CSP_CALL write, executes in kernel mode
0408 1
0409 1  Input:
0410 1      csd      address of CSD for the transfer
0411 1
0412 1  Output:
0413 1      None.
0414 1
0415 1  Routine Value:
0416 1      None.
0417 1  --
0418 1
0419 2 BEGIN                                ! Start of cluscomm_output_kernel_ast
0420 2
0421 2 LOCAL
0422 2     csd : $ref_bblock;
0423 2
0424 2
0425 2     ! If the parameter is non-zero, release that block
0426 2
0427 2 IF .csd NEQ 0
0428 2 THEN
0429 2     BEGIN
0430 2         cod = .(csd [csd$ab_data]);          ! COD address is first longword of data field
0431 2
0432 2         ! Free the CSD and put the message buffer on the list of virtual memory blocks to be deallocated
0433 2
0434 2         EXE$DEALLOC_CSD (.csd);
0435 2         collect_garbage (cod [cod_q_msgbuf]);
0436 2
0437 2         ! Place the cod in the free queue
0438 2
0439 2         $queue_remove (.cod);                ! Remove it from the queue (should be in the busy queue)
0440 2         $queue_insert_tail (.cod, cod_free_queue);
0441 2         cod_busy_count = .cod_busy_count - 1;
0442 2     END;
0443 2
0444 2     ! If we can queue another EXE$CSP_CALL, then do so
0445 2
0446 2     cod = .cod_wait_queue [0];
0447 2     WHILE .cod NEQ cod_wait_queue          ! Loop until we see the end
0448 2     AND
0449 2         .cod_busy_count LSS .cod_busy_max  ! or until we have filled our quota
0450 2 DO
0451 2     BEGIN
0452 2     LOCAL
0453 2         next,
0454 2         nod : $ref_bblock;
0455 2         next = .cod [cod_l_flink];        ! Save the pointer to the next, since we might pull it out
0456 2         nod = .cod [cod_a_nod];          ! Pointer to the nod block for the system
0457 2
0458 2     ! Make sure that the target is still there, this means that the csid stored in the node must be
```

```

: 270 0459 3 ! valid and that the node must not be in the departed state.
: 271 0460 3
: 272 0461 3 IF .nod [nod_l_node_csid] NEQ .cod [cod_l_csid] ! Node has rebooted with a new csid
: 273 0462 3 OR
: 274 0463 3 .nod [nod_b_state] EQL nod_k_state_departed ! Node is gone, but not forgotten
: 275 0464 3 THEN
: 276 0465 4 BEGIN
: 277 0466 4 $queue_remove (.cod); ! Remove it from the waiting queue
: 278 0467 4 cod [cod_l_errstat] = ss$ nodeleave; ! Give it a reasonable error status
: 279 0468 4 cod_flushed = .cod_flushed + 1; ! Count flushes individually
: 280 0469 4 cluscomm_cod_error (.cod); ! Signal and clean it up
: 281 0470 4 END
: 282 0471 3 ELSE IF cluscomm_target_in_queue (.cod, cod_busy_queue) LSS .cod_busy_node
: 283 0472 3 THEN
: 284 0473 4 BEGIN
: 285 0474 4 LOCAL
: 286 0475 4 status;
: 287 0476 4
: 288 0477 4 ! Remove the cod from the waiting queue
: 289 0478 4 $queue_remove (.cod);
: 290 0479 4
: 291 0480 4 ! Allocate a CSD block for the transfer. Common fields in the CSD are initialized by
: 292 0481 4 ! the allocate routine.
: 293 0482 4
: 294 0483 4 IF NOT (cod [cod_l_errstat] = EXE$ALLOC_CSD (csd$k_length + 4 + .cod [cod_l_msglen]; csd))
: 295 0484 5 THEN
: 296 0485 4 BEGIN
: 297 0486 5 $queue_remove (.cod); ! Signal error and clean up
: 298 0487 5 cluscomm_cod_error (.cod); ! More serious error, exit the routine
: 299 0488 5 RETURN;
: 300 0489 4 END;
: 301 0490 4 cod [cod_a_csd] = .cod; ! Point the cod at the csd
: 302 0491 4 (csd [csd$ab_data]) = .cod; ! Store cod address as first longword in csd
: 303 0492 4
: 304 0493 4 ! Set the other message dependent fields in the CSD
: 305 0494 4
: 306 0495 4 csd [csd$w_code] = csd$k_opcom; ! Set the OPCOM client code
: 307 0496 4 csd [csd$l_sendoff] = (4 + (csd [csd$ab_data])) - .cod; ! Store offset to the actual message
: 308 0497 4 csd [csd$l_sendlen] = .cod [cod_l_msglen]; ! Store size of message
: 309 0498 4 CH$MOVE (.cod [cod_l_msglen], .cod [cod_a_msgptr], ! Move the message into the CSD
: 310 0499 4 (4 + (csd [csd$ab_data]))); ! right after the cod address
: 311 0500 4 csd [csd$l_recvoff] = csd [csd$l_recvlen] = 0; ! We do not want a reply
: 312 0501 4 csd [csd$a_astadr] = cluscomm_output_kernel_ast; ! Store address of completion AST routine
: 313 0502 4 csd [csd$l_csid] = .cod [cod_l_csid]; ! Store the target node CSID
: 314 0503 5 IF NOT (cod [cod_l_errstat] = EXE$CSP_CALL (.cod [cod_a_csd]))
: 315 0504 4 THEN
: 316 0505 4 cluscomm_cod_error (.cod) ! Signal error and clean up
: 317 0506 4 ELSE
: 318 0507 5 BEGIN
: 319 0508 5 cod_busy_count = .cod_busy_count + 1; ! Bump the busy count
: 320 0509 5 cod_queued = .cod_queued + 1; ! Bump the count of those queued
: 321 0510 5 $queue_insert_tail (.cod, cod_busy_queue); ! Put it at the end of the busy queue
: 322 0511 5 $gettim (timadr=cod [cod_q_quetime]); ! Store the current time in the cod
: 323 0512 4 END;
: 324 0513 3 END;
: 325 0514 3
: 326 0515 3 ! Advance to the next one, using the saved next pointer

```

```

327      0516      3      !
328      0517      3      cod = .next;
329      0518      3      END;
330      0519      3      !
331      0520      3      ! Check the validity of the queues, crash the system if anything is wrong
332      0521      3      !
333      0522      3      %IF %VARIANT EQL 7
334      0523      3      %THEN
335      0524      3      BEGIN
336      0525      3      EXTERNAL ROUTINE monitor_queue : NOVALUE;
337      0526      3      monitor_queue (cod_busy_queue, 0);
338      0527      3      monitor_queue (cod_free_queue, 1);
339      0528      3      monitor_queue (cod_wait_queue, 2);
340      0529      3      END;
341      0530      3      %FI
342      0531      3      !
343      0532      2      RETURN;
344      0533      1      END;

```

! End of cluscomm_output_kernel_ast

-QH =
-QH =
-QH =

```

                                COD_GARBAGE_QUEUE
                                COD_FREE_QUEUE
                                COD_BUSY_QUEUE
.EXTRN EXE$DEALLOC_CSD
.EXTRN EXE$ALLOC_CSD, EXE$CSP_CALL
.EXTRN SYS$GETTIM

                                .ENTRY CLUSCOMM_OUTPUT_KERNEL_AST, Save R2,R3,R4,- ; 0402
                                R5,R6,R7,R8,R9
MOVAB COD_BUSY_COUNT, R9 ; 0427
MOVL CSD, R0
BEQL 1$
MOVL 82(R0), COD ; 0430
JSB EXE$DEALLOC_CSD ; 0434
MOVAB 32(COD), R1 ; 0435
MOVL 4(R1), R0
MOVL (R1), 8(R0)
INSQUE (R0), @_QH_+4 ; 0439
REMQUE (COD), _T_ ; 0440
INSQUE (COD), @_QH_+4 ; 0441
DECL COD_BUSY_COUNT ; 0446
MOVL COD_WAIT_QUEUE, COD ; 0447
MOVAB COD_WAIT_QUEUE, R0
CML COD, R0
BNEQ 3$
RET
                                04 A9 ; 0449
                                69 D1 0003E 3$:
                                01 19 00042
                                04 00044
                                RET
                                58 ; 0455
                                67 D0 00045 4$:
                                50 ; 0456
                                18 A7 D0 00048
                                10 A7 ; 0461
                                2C A0 D1 0004C
                                06 12 00051
                                BNEQ 5$
                                04 ; 0463
                                22 A0 91 00053
                                0E 12 00057
                                BNEQ 6$
                                50 ; 0466
                                67 OF 00059 5$:
                                REMQUE (COD), _T_

```



```

: 346 0534 1 GLOBAL ROUTINE cluscomm_cod_error (cod : $ref_bblock) : NOVALUE =
: 347 0535 1
: 348 0536 1 |**
: 349 0537 1 | Functional description:
: 350 0538 1 |
: 351 0539 1 |     This routine handles an error in CSP communications, executes in kernel mode.
: 352 0540 1 |     The error is given to a user-mode AST to actually handle
: 353 0541 1 |
: 354 0542 1 | Input:
: 355 0543 1 |     cod      address of COD for the transfer
: 356 0544 1 |
: 357 0545 1 | Output:
: 358 0546 1 |     None.
: 359 0547 1 |
: 360 0548 1 | Routine Value:
: 361 0549 1 |     None.
: 362 0550 1 | --
: 363 0551 1 |
: 364 0552 2 BEGIN                                ! Start of cluscomm_cod_error
: 365 0553 2
: 366 0554 2 LOCAL
: 367 0555 2     csd : $ref_bblock;
: 368 0556 2
: 369 0557 2     cod_errors = .cod_errors + 1;
: 370 0558 2
: 371 0559 2     Deallocate the CSD if present
: 372 0560 2
: 373 0561 2 IF (csd = .cod [cod_a_csd]) NEQ 0
: 374 0562 2 THEN
: 375 0563 2     EXE$DEALLOC_CSD (.csd);
: 376 0564 2
: 377 0565 2     Return any virtual memory to the free list
: 378 0566 2
: 379 0567 2 IF .cod [cod_l_msglen] NEQ 0
: 380 0568 2 THEN
: 381 0569 2     collect_garbage (cod [cod_q_msgbuf]);
: 382 0570 2
: 383 0571 2     Declare an AST in user mode, so that we can use RMS/etc
: 384 0572 2
: 385 0573 2 $DCLAST (astadr=cluscomm_cod_error_ast, astprm=.cod, acmode=ps($c_user));
: 386 0574 2
: 387 0575 2 RETURN;
: 388 0576 1 END;                                ! End of cluscomm_cod_error

```

QH =

COD_GARBAGE_QUEUE

		001C 00000	.ENTRY	CLUSCOMM COD_ERROR, Save R2,R3,R4	: 0534
	0000'	CF D6 00002	INCL	COD_ERRORS	: 0557
54	04	AC D0 00006	MOVL	COD, R4	: 0561
50	14	A4 D0 0000A	MOVL	20(R4), CSD	
		06 13 0000E	BEQL	1\$	
	00000000G	00 16 00010	JSB	EXE\$DEALLOC_CSD	: 0563
	20	A4 D5 00016	TSTL	32(R4)	: 0567
		11 13 00019	BEQL	2\$	

	51	20	A4	9E	0001B	MOVAB	32(R4), R1		
	50	04	A1	D0	0001F	MOVL	4(R1), R0	:	0569
08	A0		61	D0	00023	MOVL	(R1), 8(R0)	:	
0000	DF		60	0E	00027	INSQUE	(R0), @_QH_+4	:	
			03	DD	0002C	PUSHL	#3	:	0573
		04	AC	DD	0002E	PUSHL	COD	:	
		0000V	CF	9F	00031	PUSHAB	CLUSCOMM COD_ERROR_AST	:	
00000000G	00		03	FB	00035	CALLS	#3, SYSDCLAST	:	
			04	00	0003C	RET		:	0576

; Routine Size: 61 bytes, Routine Base: \$CODE\$ + 0190

```

390 0577 1 GLOBAL ROUTINE cluscomm_cod_error_ast (cod : $ref_bblock) : NOVALUE =
391 0578 1
392 0579 1 |++
393 0580 1 | Functional description:
394 0581 1 |
395 0582 1 |     This routine signals an error in CSP communications, executes in user mode.
396 0583 1 |
397 0584 1 | Input:
398 0585 1 |     cod      address of COD for the transfer
399 0586 1 |
400 0587 1 | Output:
401 0588 1 |     None.
402 0589 1 |
403 0590 1 | Routine Value:
404 0591 1 |     None.
405 0592 1 | --
406 0593 1
407 0594 2 BEGIN                                ! Start of cluscomm_cod_error
408 0595 2
409 0596 2 LOCAL
410 0597 2     leaving,
411 0598 2     dsc : VECTOR [2, LONG],
412 0599 2     nod : $ref_bblock;
413 0600 2
414 0601 2 nod = .cod [cod_a_nod];
415 0602 2 leaving = (.cod [cod_l_errstat] EQL ss$_nodeleave);
416 0603 3 IF (NOT .leaving)                       ! If any other error
417 0604 3     OR
418 0605 3     (NOT .nod [nod_v_node_leaving])     ! or if the first node_leaving error
419 0606 3 THEN
420 0607 3     BEGIN
421 0608 3     |
422 0609 3     | Put a message in the logfile
423 0610 3     |
424 0611 3     | clusutil_node_message (.nod, opc$_cluscomm, false);
425 0612 3     |
426 0613 3     | If any error besides leaving, then put a message in the logfile about the exact reason
427 0614 3     |
428 0615 3     | IF .leaving                           ! Mark the first message so that we can skip the others
429 0616 3     | THEN
430 0617 3     |     nod [nod_v_node_leaving] = true
431 0618 3     | ELSE
432 0619 4     | BEGIN
433 0620 4     |     write_log_file (
434 0621 4     |         share_fao_buffer (%ASCID %STRING ('Unable to communicate with !AS (!XL), system status code !XL!
435 0622 4     |             Current statistics for cluster message activity:!/
436 0623 4     |             Msg desc allocated!8UL      Errors      !8UL!/'
437 0624 4     |             Msg requests      !8UL      Msgs flushed !8UL!/'
438 0625 4     |             Msgs queued       !8UL      Current busy !8UL'),
439 0626 4     |         nod [nod_q_name_desc], .cod [cod_l_csid], .cod [cod_l_errstat],
440 0627 4     |         .cod_allocated, .cod_errors-.cod_flushed,
441 0628 4     |         .cod_requests, .cod_flushed,
442 0629 4     |         .cod_queued, .cod_busy_count));
443 0630 4     |
444 0631 4     | Write some more arcane, but useful messages if we are debugging
445 0632 4     |
446 0633 4     %IF %VARIANT NEQ 0

```



```

: 447      U 0634 4      %THEN
448      U 0635 4      dsc [0] = cod_k_size; dsc [1] = .cod;
449      U 0636 4      dump_log_file (dsc, %ASCII 'Dump of COD used in transfer');
450      U 0637 4      dump_log_file (cod [cod_q_msgbuf], %ASCII 'Dump of COD text field');
451      0638 4      %FI
452      0639 3      END;
453      0640 2      END;
454      0641 2
455      0642 2 $queue_insert_tail (.cod, cod_free_queue);      . All done, put it back in the queue
456      0643 2
457      0644 2 RETURN;
: 458      0645 1 END;      ! End of cluscomm_cod_error_ast

```

```

.PSECT $SPLITS,NOWRT,NOEXE,2
75 6D 6D 6F 63 20 6F 74 20 65 6C 62 61 6E 55 00000 P.AAB: .ASCII \Unable to communicate with !AS (!XL), sy\
53 41 21 20 68 74 69 77 20 65 74 61 63 69 6E 0000F
   79 73 20 2C 29 4C 58 21 28 20 0001E
64 6F 63 20 73 75 74 61 74 73 20 6D 65 74 73 00028 .ASCII \stem status code !XL!/ Current statisti\
6E 65 72 72 75 43 20 20 2F 21 4C 58 21 20 65 00037
   69 74 73 69 74 61 74 73 20 74 00046
20 72 65 74 73 75 6C 63 20 72 6F 66 20 73 63 00050 .ASCII \cs for cluster message activity:!\<9>
74 69 76 69 74 63 61 20 65 67 61 73 73 65 6D 0005F
   09 2F 21 3A 79 0006E
61 63 6F 6C 6C 61 20 63 73 65 64 20 67 73 4D 00073 .ASCII \Msg desc allocated!8UL      Errors      \
72 72 45 20 20 20 20 20 4C 55 38 21 64 65 74 00082
   20 20 20 20 20 20 20 73 72 6F 00091
75 71 65 72 20 67 73 4D 09 2F 21 4C 55 38 21 0009B .ASCII \!8UL!\<9>\Msg requests      !8UL      \
20 4C 55 38 21 20 20 20 20 20 20 73 74 73 65 000AA
   20 20 20 20 20 20 20 20 20 20 000B9
38 21 20 64 65 68 73 75 6C 66 20 73 67 73 4D 000BD .ASCII \Msgs flushed !8UL!\<9>\Msgs queued      \
65 75 65 75 71 20 73 67 73 4D 09 2F 21 4C 55 000CC
   20 20 20 64 000DB
75 43 20 20 20 20 20 4C 55 38 21 20 20 20 20 000DF .ASCII \      !8UL      Current busy !8UL\<0><0>
4C 55 38 21 20 79 73 75 62 20 74 6E 65 72 72 000EE
   00 00 000FD
   00 000FF
010E00FD 00100 P.AAA: .ASCII <0>
00000000 00104 .LONG 17694973
          .ADDRESS P.AAB

```

QH = COD_FREE_QUEUE

```

.PSECT $CODE$,NOWRT,2
          003C 00000 .ENTRY CLUSCOMM_COD_ERROR_AST, Save R2,R3,R4,R5      : 0577
55 0000' CF 9E 00002 MOVAB COD_FLUSHED, R5
5E 08 C2 00007 SUBL2 #8, SP
53 04 AC D0 0000A MOVL COD, R3      : 0601
52 18 A3 D0 0000E MOVL 24(R3), NOD
          50 D4 00012 CLRL R0      : 0602
0000223C 8F 1C A3 D1 00014 CMPL 28(R3), #8764
          02 12 0001C BNEQ 1$
          50 D6 0001E INCL R0
          54 50 D0 00020 1$: MOVL R0, LEAVING

```

44	2A	05 A2		54 E9 00023	BLBC	LEAVING, 2\$: 0603
				03 E0 00026	BBS	#3, 42(NOD), 4\$: 0605
				7E D4 0002B	CLRL	-(SP)	: 0611
		00058253		8F DD 0002D	PUSHL	#361043	
				52 DD 00033	PUSHL	NOD	
	0000G	CF		03 FB 00035	CALLS	#3, CLUSUTIL_NODE_MESSAGE	
		06		54 E9 0003A	BLBC	LEAVING, 3\$: 0615
	2A	A2		08 88 0003D	BISB2	#8, 42(NOD)	: 0617
				2C 11 00041	BRB	4\$	
			F0	A5 DD 00043	PUSHL	COD_BUSY_COUNT	: 0629
			08	A5 DD 00046	PUSHL	COD_QUEUED	
				65 DD 00049	PUSHL	COD_FLUSHED	: 0628
			04	A5 DD 0004B	PUSHL	COD_REQUESTS	
7E	FC	A5		65 C3 0004E	SUBL3	COD_FLUSHED, COD_ERRORS, -(SP)	: 0627
				65 A5 DD 00053	PUSHL	COD_ALLOCATED	
			EC	A5 DD 00056	PUSHL	28(R3)	: 0626
			1C	A3 DD 00059	PUSHL	16(R3)	
			10	A3 DD 0005F	PUSHAB	48(NOD)	
			30	A2 9F 0005C	PUSHAB	P.AAA	: 0625
			0000'	CF 9F 0005F	PUSHAB	P.AAA	: 0626
	0000G	CF		0A FB 00063	CALLS	#10, SHARE_FAO_BUFFER	
				50 DD 00068	PUSHL	R0	
	0000G	CF		01 FB 0006A	CALLS	#1, WRITE_LOG_FILE	
	18	B5		63 0E 0006F	INSQUE	(R3), @_QR_+4	: 0642
				04 00073	RET		: 0645

; Routine Size: 116 bytes. Routine Base: \$CODE\$ + 01CD

```

0646 1 GLOBAL ROUTINE CLUSCOMM_SEND (CSID, MSG_LEN, MSG_PTR) = %SBTTL 'CLUSCOMM_SEND (CSID, MSG_LEN, MSG_PT
0647 1
0648 1 !++
0649 1 Functional description:
0650 1
0651 1 Jacket routine to send a message to remote node(s), and wait for completion.
0652 1
0653 1 Input:
0654 1
0655 1 CSID - Id of target node, -1 for broadcast to all nodes except local
0656 1 MSG_LEN - Length of message
0657 1 MSG_PTR - Address of message
0658 1
0659 1 Implicit Input:
0660 1
0661 1 None.
0662 1
0663 1 Output:
0664 1
0665 1 None.
0666 1
0667 1 Implicit output:
0668 1
0669 1 None.
0670 1
0671 1 Side effects:
0672 1
0673 1 Messages will be sent to remote nodes.
0674 1
0675 1 Routine value:
0676 1
0677 1 Status from comm primitive.
0678 1 --
0679 1
0680 2 BEGIN ! Start of CLUSCOMM_SEND
0681 2
0682 2 EXTERNAL
0683 2 GLOBAL_STATUS : BITVECTOR [32],
0684 2 LCL_CSID : LONG,
0685 2 NOD_HEAD : VECTOR [2, LONG];
0686 2
0687 2 LOCAL
0688 2 FINAL_STAT : LONG,
0689 2 NOD : $ref_block,
0690 2 STATUS : LONG;

```

```

506 0691 2 |
507 0692 2 | Assume success for final status
508 0693 2 |
509 0694 2 | FINAL_STAT = SS$_NORMAL;
510 0695 2 |
511 0696 2 | If not in a cluster we are done, return with success
512 0697 2 |
513 0698 2 | IF NOT .GLOBAL_STATUS [GBLSTS_K_IN_VAXcluster]
514 0699 2 | THEN
515 0700 2 |     RETURN .FINAL_STAT;
516 0701 2 |
517 0702 2 | If CSID is -1, send it to everyone
518 0703 2 |
519 0704 2 | IF .CSID EQL -1
520 0705 2 | THEN
521 0706 3 |     BEGIN
522 0707 3 |     NOD = .NOD_HEAD [0];
523 0708 3 |     WHILE .NOD NEQ NOD_HEAD [0]
524 0709 3 |     DO
525 0710 4 |         BEGIN
526 0711 4 |         LOCAL
527 0712 4 |         TARGET;
528 0713 4 |         |
529 0714 4 |         | Send to all nodes but local
530 0715 4 |         |
531 0716 4 |         TARGET = .NOD [NOD_L_NODE_CSID];
532 0717 4 |         IF .TARGET NEQ .LCL_CSID
533 0718 4 |         THEN
534 0719 5 |             BEGIN
535 0720 5 |             STATUS = CLUSCOMM_SEND_ONE (.TARGET, .NOD, .MSG_LEN, .MSG_PTR);
536 0721 5 |             IF NOT .STATUS
537 0722 5 |             THEN
538 0723 5 |                 FINAL_STAT = .STATUS;
539 0724 4 |             END;
540 0725 4 |         |
541 0726 4 |         | Move to the next node
542 0727 4 |         |
543 0728 4 |         NOD = .NOD [NOD_L_FLINK];
544 0729 3 |         END;
545 0730 3 |     END
546 0731 3 |
547 0732 3 | CSID is real, send it to a single node
548 0733 3 |
549 0734 2 | ELSE
550 0735 3 |     BEGIN
551 0736 3 |     NOD = CLUSUTIL_FIND_NOD_BY_CSID (.CSID);
552 0737 4 |     FINAL_STAT = (IF .NOD EQL 0
553 0738 4 |                 THEN SS$_NOSUCHNODE
554 0739 3 |                 ELSE CLUSCOMM_SEND_ONE (.CSID, .NOD, .MSG_LEN, .MSG_PTR));
555 0740 2 |     END;
556 0741 2 |
557 0742 2 | RETURN .FINAL_STAT;
558 0743 1 | END;

```

! End of CLUSCOMM_SEND

.EXTRN GLOBAL_STATUS, LCL_CSID

				.EXTRN	NOD_HEAD	
			000C 00000	.ENTRY	CLUSCOMM_SEND, Save R2,R3	: 0646
	53		01 D0 00002	MOVL	#1, FINAL_STAT	: 0694
	5F	0000G	CF E9 00005	BLBC	GLOBAL_STATUS+1, 5\$: 0698
FFFFFFFF	8F	04	AC D1 0000A	CMPL	CSID, #1	: 0704
			30 12 00012	BNEQ	3\$	
	52	0000G	CF D0 00014	MOVL	NOD_HEAD, NOD	: 0707
	51	0000G	CF 9E 00019	MOVAB	NOD_HEAD, R1	: 0708
	51		52 D1 0001E	CMPL	NOD, R1	
			46 13 00021	BEQL	5\$	
	51	2C	A2 D0 00023	MOVL	44(NOD), TARGET	: 0716
0000G	CF		51 D1 00027	CMPL	TARGET, LCL_CSID	: 0717
			11 13 0002C	BEQL	2\$	
	7E	08	AC 7D 0002E	MOVQ	MSG_LEN, -(SP)	: 0720
			06 BB 00032	PUSHR	#*MZR1,R2>	
0000V	CF		04 FB 00034	CALLS	#4, CLUSCOMM_SEND_ONE	
	03		50 E8 00039	BLBS	STATUS, 2\$: 0721
	53		50 D0 0003C	MOVL	STATUS, FINAL_STAT	: 0723
	52		62 D0 0003F	MOVL	(NOD), NOD	: 0728
			D5 11 00042	BRB	1\$: 0708
		04	AC DD 00044	PUSHL	CSID	: 0736
0000G	CF		01 FB 00047	CALLS	#1, CLUSUTIL_FIND_NOD_BY_CSID	
	52		50 D0 0004C	MOVL	R0, NOD	
			07 12 0004F	BNEQ	4\$: 0737
	53	028C	8F 3C 00051	MOVZWL	#652, FINAL_STAT	
			11 11 00056	BRB	5\$	
	7E	08	AC 7D 00058	MOVQ	MSG_LEN, -(SP)	: 0739
			52 DD 0005C	PUSHL	NOD	
		04	AC DD 0005E	PUSHL	CSID	
0000V	CF		04 FB 00061	CALLS	#4, CLUSCOMM_SEND_ONE	
	53		50 D0 00066	MOVL	R0, FINAL_STAT	
	50		53 D0 00069	MOVL	FINAL_STAT, R0	: 0742
			04 0006C	RET		: 0743

; Routine Size: 109 bytes, Routine Base: \$CODE\$ + 0241

```

: 560 0744 1 GLOBAL ROUTINE CLUSCOMM_SEND_ONE (CSID, NOD, MSG_LEN, MSG_PTR) = %SBTTL 'CLUSCOMM_SEND_ONE'
: 561 0745 1
: 562 0746 1 |++
: 563 0747 1 | Functional description:
: 564 0748 1 |
: 565 0749 1 |     Send a message to a remote node, and wait for completion.
: 566 0750 1 |
: 567 0751 1 | Input:
: 568 0752 1 |
: 569 0753 1 |     CSID    - Id of target node
: 570 0754 1 |     NOD     - Address of NOD block for target node
: 571 0755 1 |     MSG_LEN - Length of message
: 572 0756 1 |     MSG_PTR - Address of message
: 573 0757 1 |
: 574 0758 1 | Implicit Input:
: 575 0759 1 |
: 576 0760 1 |     None.
: 577 0761 1 |
: 578 0762 1 | Output:
: 579 0763 1 |
: 580 0764 1 |     None.
: 581 0765 1 |
: 582 0766 1 | Implicit output:
: 583 0767 1 |
: 584 0768 1 |     None.
: 585 0769 1 |
: 586 0770 1 | Side effects:
: 587 0771 1 |
: 588 0772 1 |     Messages will be sent to remote nodes.
: 589 0773 1 |
: 590 0774 1 | Routine value:
: 591 0775 1 |
: 592 0776 1 |     Status from comm primitive.
: 593 0777 1 | --
: 594 0778 1 |
: 595 0779 2 BEGIN                                ! Start of CLUSCOMM_SEND_ONE
: 596 0780 2
: 597 0781 2 EXTERNAL
: 598 0782 2     GLOBAL_STATUS    : BITVECTOR [32];
: 599 0783 2
: 600 0784 2 LOCAL
: 601 0785 2     ARGLIST           : VECTOR [2, LONG],
: 602 0786 2     COD               : $ref_bblock,
: 603 0787 2     STATUS           : LONG;
: 604 0788 2
: 605 0789 2 |
: 606 0790 2 | If not in a cluster we are done, return with error.
: 607 0791 2 |
: 608 0792 2 IF NOT .GLOBAL_STATUS [GBLSTS_K_IN_VAXcluster]
: 609 0793 2 THEN
: 610 0794 2     RETURN SS$_NOSUCHNODE;
: 611 0795 2 |
: 612 0796 2 | Allocate and fill in the COD
: 613 0797 2 |
: 614 0798 2 COD = CLUSCOMM COD_ALLOCATE ();           ! Get a new COD
: 615 0799 2 COD [COD_L_CSID] = .CSID;                ! Keep a copy of the CSID in the COD
: 616 0800 2 COD [COD_A_NOD] = .NOD;                  ! Keep the NOD address too

```


	50		OC	AC	D0	00025		MOVL	MSG_LEN, R0		0801
	OC			50	D1	00029		CMPL	R0, #12		
				03	1E	0002C		BGEQU	2\$		
	20	50		OC	D0	0002E		MOVL	#12, R0		
		A6		50	D0	00031	2\$:	MOVL	R0, 32(R6)		
				24	A6	9F	00035	PUSHAB	36(R6)		0802
				20	A6	9F	00038	PUSHAB	32(R6)		
	0000G	CF		02	FB	0003B		CALLS	#2, OPC\$GET_VM		
		57		50	D0	00040		MOVL	R0, STATUS		
		28		57	E9	00043		BLBC	STATUS, 3\$		
24	B6	10		OC	AC	28	00046	MOVC3	MSG_LEN, @MSG_PTR, @36(R6)		0805
		0000'			66	0E	0004D	INSQUE	(R6), @QH +4		0809
				0000'	CF	D6	00052	INCL	COD_REQUESTS		0810
					5E	DD	00056	PUSHL	SP		0815
				FD70	CF	9F	00058	PUSHAB	CLUSCOMM_DECLARE_KERNEL_AST		
	00000000G	00			02	FB	0005C	CALLS	#2, SYS\$CMKRNL		
		57			50	D0	00063	MOVL	R0, STATUS		
		18			57	E8	00066	BLBS	STATUS, 5\$		0820
		1C			57	D1	00069	CMPL	STATUS, #28		0823
					0A	13	0006C	BEQL	4\$		
					57	DD	0006E	3\$:	PUSHL	STATUS	0825
	000C0000G	00			01	FB	00070	CALLS	#1, LIB\$STOP		
						04	00077	RET			
				0000'	CF	9F	00078	4\$:	PUSHAB	P.AAC	0827
	0000G	CF			01	FB	0007C	CALLS	#1, WRITE_LOG_FILE		
		50			57	D0	00081	5\$:	MOVL	STATUS, R0	0830
					04	00084		RET			0831

; Routine Size: 133 bytes. Routine Base: \$CODE\$ + 02AE


```

: 649 0832 1 GLOBAL ROUTINE cluscomm_target_in_queue (cod : $ref_bblock, queue : $ref_bblock) =
: 650 0833 1
: 651 0834 1 !++
: 652 0835 1 | Functional description:
: 653 0836 1 |
: 654 0837 1 |     Check to see if the CSID field in the cod is in any of the CODs in the queue.
: 655 0838 1 |     We assume that we are operating at AST level so that we do not have to worry
: 656 0839 1 |     about interlocking the queue.
: 657 0840 1 |
: 658 0841 1 | Input:
: 659 0842 1 |     cod     pointer to a cod
: 660 0843 1 |     queue   head of a queue of CODs
: 661 0844 1 |
: 662 0845 1 | Output:
: 663 0846 1 |     None.
: 664 0847 1 |
: 665 0848 1 | Routine Value:
: 666 0849 1 |     number of matches in the queue
: 667 0850 1 | --
: 668 0851 1
: 669 0852 2 BEGIN                               ! Start of cluscomm_TARGET_IN_QUEUE
: 670 0853 2
: 671 0854 2 LOCAL
: 672 0855 2     count,
: 673 0856 2     csid,
: 674 0857 2     head : $ref_bblock,
: 675 0858 2     cur  : $ref_bblock;
: 676 0859 2
: 677 0860 2 |
: 678 0861 2 | Scan the queue, counting the number of times the target appears
: 679 0862 2 |
: 680 0863 2 count = 0;
: 681 0864 2 csid = .cod [cod_l_csid];
: 682 0865 2 head = .queue;
: 683 0866 2 cur = .head [cod_l_flink];
: 684 0867 2 WHILE .cur NEQ .head                       ! Loop until we see the end
: 685 0868 2 DO
: 686 0869 2     BEGIN
: 687 0870 2     IF .csid EQL .cur [cod_l_csid]
: 688 0871 2     THEN
: 689 0872 2         count = .count + 1;
: 690 0873 2         cur = .cur [cod_l_flink];           ! Get the next cod
: 691 0874 2     END;
: 692 0875 2
: 693 0876 2 RETURN .count;
: 694 0877 1 END;                               ! End of cluscomm_TARGET_IN_QUEUE

```

		000C 00000	.ENTRY	CLUSCOMM_TARGET_IN_QUEUE, Save R2,R3	: 0832
		52 D4 00002	CLRL	COUNT	: 0863
50	04	AC 7D 00004	MOVQ	COD, R0	: 0864
53	10	A0 D0 00008	MOVL	16(R0), CSID	:
50		61 D0 0000C	MOVL	(HEAD), CUR	: 0866
51		50 D1 0000F 18:	CMPL	CUR, HEAD	: 0867

OPC\$CLUSCOMM
V04-000

CLUSCOMM_SEND_ONE

K 4
16-Sep-1984 01:20:02
14-Sep-1984 12:50:36

VAX-11 Bliss-32 V4.0-742
[OPCOM.SRC]CLUSCOMM.B32;1

Page 24
(11)

10	A0	0D	13	00012	BEQL	3\$:	
		53	D1	00014	CMPL	(SID, 16(CUR)	:	0870
		02	12	00018	BNEQ	2\$:	
		52	D6	0001A	INCL	COUNT	:	0872
	50	60	D0	0001C	2\$:	MOVL	:	0873
		EE	11	0001F	BRB	1\$:	0867
	50	52	D0	00021	3\$:	MOVL	:	0876
		04	00024		RET		:	0877

; Routine Size: 37 bytes, Routine Base: \$CODE\$ + 0333

: 696 0878 1 END
: 697 0879 0 ELUDOM

. End of CLUSCOMM

PSECT SUMMARY

Name	Bytes	Attributes
\$GLOBALS	64	NOVEC, WRT, RD, NOEXE, NOSHR, LCL, REL, CON, NOPIC, ALIGN(2)
\$CODES	856	NOVEC, NOWRT, RD, EXE, NOSHR, LCL, REL, CON, NOPIC, ALIGN(2)
\$SPLITS	312	NOVEC, NOWRT, RD, NOEXE, NOSHR, LCL, REL, CON, NOPIC, ALIGN(2)

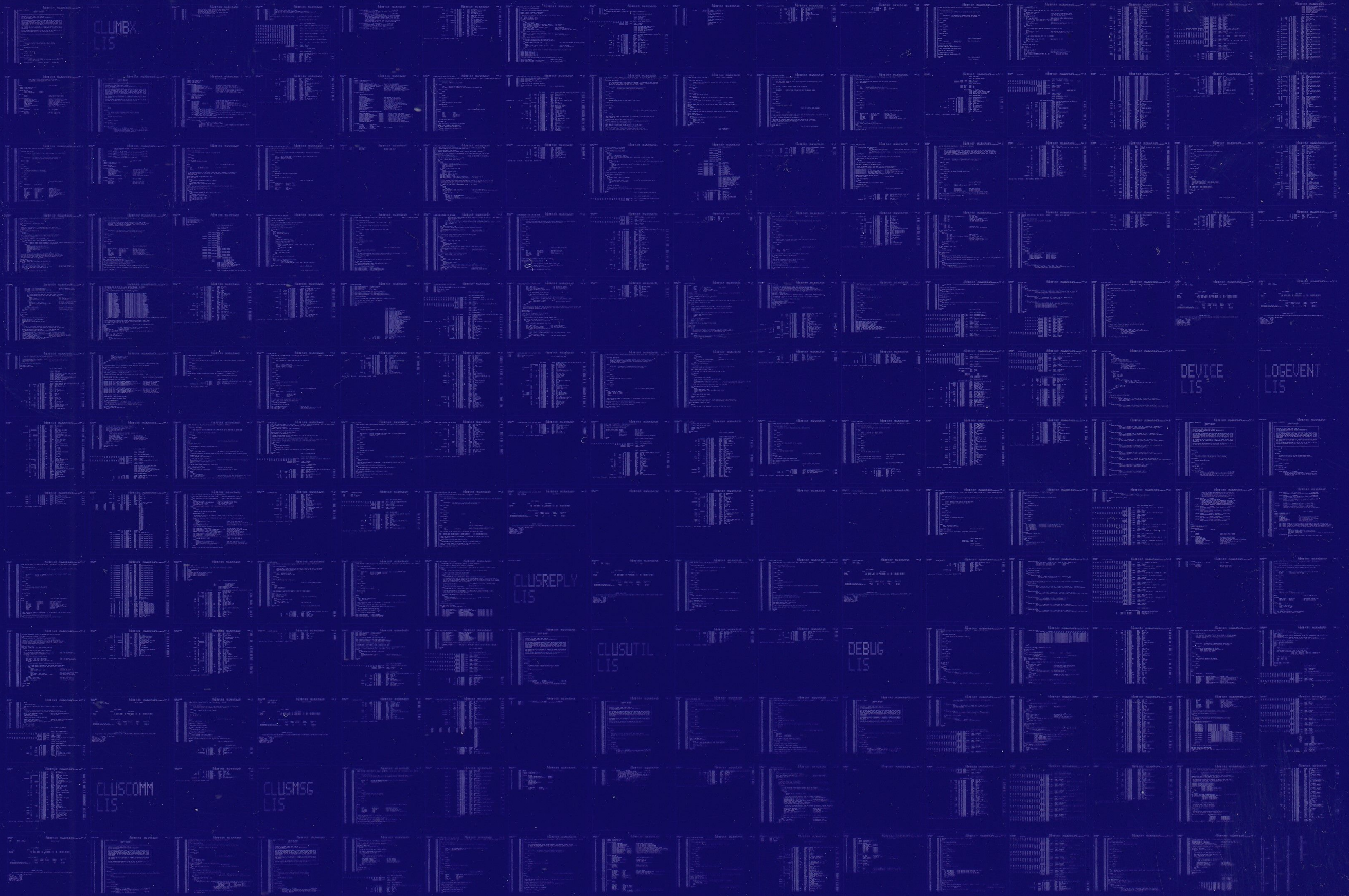
Library Statistics

File	Total	Symbols Loaded	Percent	Pages Mapped	Processing Time
\$_\$255\$DUA28:[SYSLIB]LIB.L32;1	18619	12	0	1000	00:01.8
\$_\$255\$DUA28:[OPCOM.OBJ]OPCOMLIB.L32;1	633	34	5	43	00:00.9

COMMAND QUALIFIERS

BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/LIS=LIS\$:CLUSCOMM/OBJ=OBJ\$:CLUSCOMM MSRC\$:CLUSCOMM/UPDATE=(ENH\$:CLUSCOMM)

: Size: 856 code + 376 data bytes
 : Run Time: 00:22.2
 : Elapsed Time: 01:18.8
 : Lines/CPU Min: 2375
 : Lexemes/CPU-Min: 22248
 : Memory Used: 161 pages
 : Compilation Complete



DEVICE LIS
LOGEVENT LIS

CLUSREPLY LIS

CLUSUTIL LIS

DEBUG LIS

CLUSCOMM LIS

CLUSMSG LIS