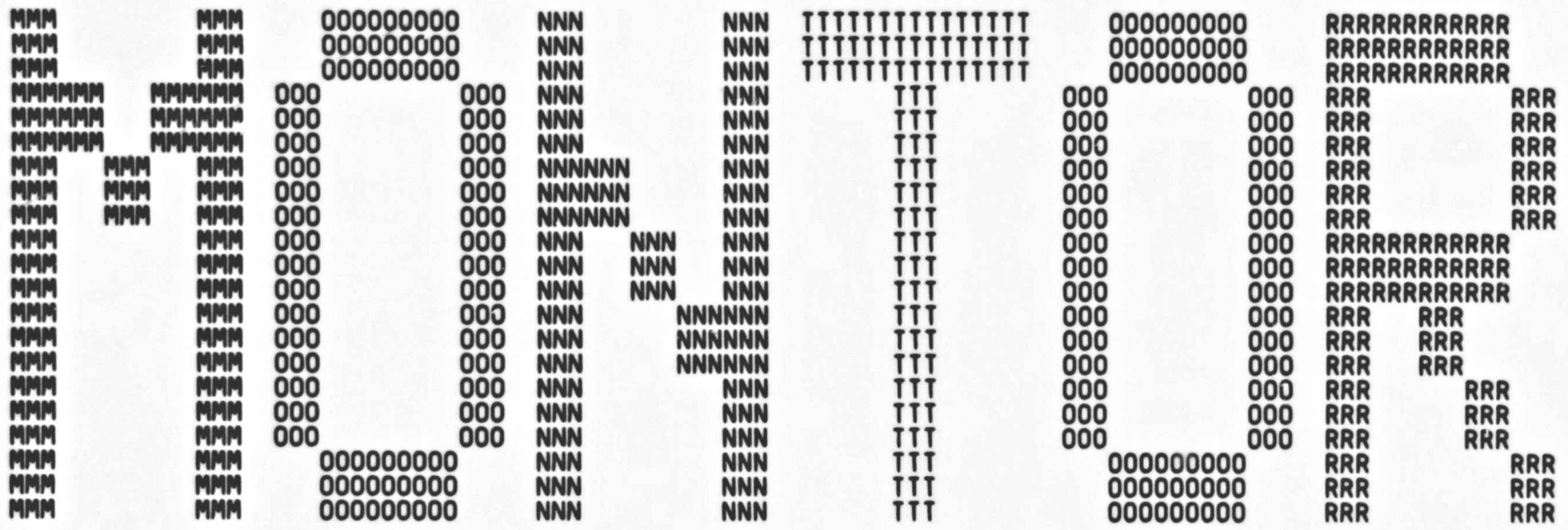
MMM MMM 000000000 NNN NNN TTTTTTTTTTTTTTT 000000000 RRRRRRRRRRR
MMM MMM 000000000 NNN NNN TTTTTTTTTTTTTTT 000000000 RRRRRRRRRRRR
MMM MMM 000000000 NNN NNN TTTTTTTTTTTTTTT 000000000 RRRRRRRRRRRR
MMMMMMM MMMMMMM 000 000 NNN NNN TTT 000 000 RRR RRR
MMMMMMM MMMMMMM 000 000 NNN NNN TTT 000 000 RRR RRR
MMMMMMM MMMMMMM 000 000 NNN NNN TTT 000 000 RRR RRR
MMM MMM MMM 000 000 NNNNNN NNN TTT 000 000 RRR RRR
MMM MMM MMM 000 000 NNNNNN NNN TTT 000 000 RRR RRR
MMM MMM MMM 000 000 NNNNNN NNN TTT 000 000 RRR RRR
MMM MMM 000 000 NNN NNN NNN TTT 000 000 RRRRRRRRRRR
MMM MMM 000 000 NNN NNN NNN TTT 000 000 RRRRRRRRRRR
MMM MMM 000 000 NNN NNN NNN TTT 000 000 RRRRRRRRRRR
MMM MMM 000 000 NNN NNNNN TTT 000 000 RRR RRR
MMM MMM 000 000 NNN NNNNN TTT 000 000 RRR RRR
MMM MMM 000 000 NNN NNNNN TTT 000 000 RRR RRR
MMM MMM 000 000 NNN NNN TTT 000 000 RRR RRR
MMM MMM 000 000 NNN NNN TTT 000 000 RRR RRR
MMM MMM 000 000 NNN NNN TTT 000 000 RRR RRR
MMM MMM 000000000 NNN NNN TTT 000000000 RRR RRR
MMM MMM 000000000 NNN NNN TTT 000000000 RRR RRR
MMM MMM 000000000 NNN NNN TTT 000000000 RRR RRR

FILEID**COLLEVT

```
  CCCCCCCC    000000    LL          LL          EEEEEEEEEE  VV      VV  TTTTTTTTTT
  CCCCCCCC    000000    LL          LL          EEEEEEEEEE  VV      VV  TTTTTTTTTT
CC          OO      OO  LL          LL          EE          VV      VV      TT
CC          OO      OO  LL          LL          EE          VV      VV      TT
CC          OO      OO  LL          LL          EE          VV      VV      TT
CC          OO      OO  LL          LL          EE          VV      VV      TT
CC          OO      OO  LL          LL          EEEEEEEE    VV      VV      TT
CC          OO      OO  LL          LL          EEEEEEEE    VV      VV      TT
CC          OO      OO  LL          LL          EE          VV      VV      TT
CC          OO      OO  LL          LL          EE          VV      VV      TT
CC          OO      OO  LL          LL          EE            VV  VV        TT          ....
CC          OO      OO  LL          LL          EE            VV  VV        TT          ....
  CCCCCCCC    000000    LLLLLLLLLL  LLLLLLLLLL  EEEEEEEEEE      VV          TT          ....
  CCCCCCCC    000000    LLLLLLLLLL  LLLLLLLLLL  EEEEEEEEEE      VV          TT          ....

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
```

```
1          COLLECTION_EVENT: Procedure     Options(Ident('V04-000'));
2     1
3     1    /*
4     1    /*****************************************************************************
5     1    /*                                                                           *
6     1    /*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                  *
7     1    /*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                   *
8     1    /*  ALL RIGHTS RESERVED.                                                     *
9     1    /*                                                                           *
10    1    /*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED    *
11    1    /*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE    *
12    1    /*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER     *
13    1    /*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY    *
14    1    /*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY    *
15    1    /*  TRANSFERRED.                                                             *
16    1    /*                                                                           *
17    1    /*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE    *
18    1    /*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT    *
19    1    /*  CORPORATION.                                                             *
20    1    /*                                                                           *
21    1    /*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS    *
22    1    /*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                  *
23    1    /*                                                                           *
24    1    /*                                                                           *
25    1    /*****************************************************************************
26    1    /*/
27    1
28    1    /*
29    1    /*++
30    1    /* FACILITY:  MONITOR Utility
31    1    /*
32    1    /* ABSTRACT:  COLLECTION_EVENT AST Routine.
33    1    /*
34    1    /*              Queued from the EXECUTE_REQUEST routine each time a
35    1    /*                data collection is required.
36    1    /*
37    1    /*
38    1    /* ENVIRONMENT:
39    1    /*
40    1    /*      VAX/VMS operating system, unprivileged user mode,
41    1    /*               except for certain collection routines which
42    1    /*               run in EXEC or KERNEL mode to access system
43    1    /*               data bases.
44    1    /*
45    1    /* AUTHOR:  Thomas L. Cafarella, April, 1981
46    1    /*
47    1
```

```
48  1       /*
49  1       /* MODIFIED BY:
50  1       /*
51  1       /*      V03-015 TLC1085         Thomas L. Cafarella     22-Jul-1984     14:00
52  1       /*              Calculate scale values for Free and Modified List bar graphs.
53  1       /*
54  1       /*      V03-014 TLC1082         Thomas L. Cafarella     23-Jul-1984     11:00
55  1       /*              Force error message when playing back a file containing
56  1       /*                      only one collection.
57  1       /*
58  1       /*      V03-013 TLC1072         Thomas L. Cafarella     17-Apr-1984     11:00
59  1       /*              Add volume name to DISK display.
60  1       /*
61  1       /*      V03-012 TLC1066         Thomas L. Cafarella     01-Apr-1984     11:00
62  1       /*              Add SYSTEM class.
63  1       /*
64  1       /*      V03-011 TLC1061         Thomas L. Cafarella     18-Mar-1984     11:00
65  1       /*              Identify dual-path disks by allocation class.
66  1       /*
67  1       /*      V03-011 TLC1058         Thomas L. Cafarella     23-Mar-1984     10:00
68  1       /*              Fix MODES class when 782 and non-782 input
69  1       /*              files mixed in multi-file summary.
70  1       /*
71  1       /*      V03-010 TLC1051         Thomas L. Cafarella     11-Jan-1984     11:00
72  1       /*              Add consecutive number to class header record.
73  1       /*
74  1       /*      V03-010 PRS1002         Paul R. Senn            29-Dec-1983     16:00
75  1       /*              GLOBALDEF VALUE symbols must now be longwords;
76  1       /*              Use %REPLACE rather than GLOBALDEF VALUE for any equated
77  1       /*              symbols which are not 4 bytes in length;
78  1       /*
79  1       /*      V03-009 TLC1050         Thomas L. Cafarella     06-Dec-1983     11:00
80  1       /*              Change directory information in DLOCK class.
81  1       /*
82  1       /*      V03-008 TLC1046         Thomas L. Cafarella     26-Aug-1983     18:00
83  1       /*              Force flush to occur after all classes written to file.
84  1       /*
85  1       /*      V03-007 TLC1040         Thomas L. Cafarella     15-Jun-1983     10:00
86  1       /*              Add directory node indicator to DLOCK class.
87  1       /*
88  1       /*      V03-006 TLC1035         Thomas L. Cafarella     06-Jun-1983     15:00
89  1       /*              Add homogeneous class type and DISK class.
90  1       /*
91  1       /*      V03-005 TLC1029         Thomas L. Cafarella     21-Apr-1983     10:00
92  1       /*              Correctly calculate "Interrupt Stack" string.
93  1       /*
94  1       /*      V03-004 TLC1028         Thomas L. Cafarella     14-Apr-1983     16:00
95  1       /*              Add interactive user interface.
96  1       /*
97  1       /*      V03-001 TLC0014         Thomas L. Cafarella     01-Apr-1982     13:00
98  1       /*              Correct attached processor time reporting for MODES.
99  1       /*
100 1       /*      V03-003 TLC1011         Thomas L. Cafarella     29-Mar-1982     20:00
101 1       /*              Move system service names for SSERROR msg to static storage.
102 1       /*
103 1       /*      V03-002 TLC1003         Thomas L. Cafarella     23-Mar-1982     13:00
```

```
  104 ¦  1        /*              Fix up module headers.
  105 ¦  1        /*
  106 ¦  1        /*      V03-001 TLC1002         Thomas L. Cafarella     20-Mar-1982     13:00
  107 ¦  1        /*              Move collection event flag to REQUEST.PLI for consolidation.
  108 ¦  1        /*
  109 ¦  1        /*              Compress bar graph range for MODES and TOPCPU.
  110 ¦  1        /*
  111 ¦  1        /*--
  112 ¦  1        /*/
  113    1
```

```
114  1      /*
115  1      /*  +-------------------------------------------------------------------+
116  1      /*  |                                                                   |
117  1      /*  |                        INCLUDE  FILES                             |
118  1      /*  |                                                                   |
119  1      /*  +-------------------------------------------------------------------+
120  1      /*/
121  1
122  1      %INCLUDE        MONDEF;                         /* Monitor utility structure definitions */
890  1
891  1      /*
892  1      /*  +-------------------------------------------------------------------+
893  1      /*  |                                                                   |
894  1      /*  |                SYSTEM SERVICE MACRO DEFINITIONS                   |
895  1      /*  |                                                                   |
896  1      /*  +-------------------------------------------------------------------+
897  1      /*/
898  1
899  1      %INCLUDE        SYS$SETIMR;                     /* $SETIMR system service */
908  1
```

```
909  1      /*
910  1      /*      +-----------------------------------------------------------------------+
911  1      /*      |                                                                       |
912  1      /*      |                  EXTERNAL STORAGE  DEFINITIONS                        |
913  1      /*      |                                                                       |
914  1      /*      +-----------------------------------------------------------------------+
915  1      /*/
916  1
917  1      Declare
918  1      MNR$_CLASMISS    FIXED BINARY(31) GLOBALREF VALUE,      /* Error message */
919  1      MNR$_SSERROR     FIXED BINARY(31) GLOBALREF VALUE,      /* Error message */
920  1      MNR$_BEGNLEND    FIXED BINARY(31) GLOBALREF VALUE,      /* Error message */
921  1      MNR$_COLLERR     FIXED BINARY(31) GLOBALREF VALUE;      /* Error message */
922  1
923  1      Declare
924  1      COLL_EV_FLAG     FIXED BINARY(31) GLOBALREF VALUE,      /* Collection event flag */
925  1      MAX_CLASS_NO     FIXED BINARY(31) GLOBALREF VALUE,      /* Maximum defined class number */
926  1      SKIP_TO_CLASS    FIXED BINARY(31) GLOBALREF VALUE;      /* Skip to class record indicator for READ_INPUT rtn */
927  1
928  1      Declare
929  1      COLLENDED        BIT(1) ALIGNED   GLOBALREF,            /* YES => collection has ended */
930  1      COLL_STATUS      FIXED BINARY(31) GLOBALREF,            /* COLLECTION_EVENT return status code */
931  1      NORMAL           FIXED BINARY(31) GLOBALREF,            /* MONITOR normal return status */
932  1      MULT_TEMP        FIXED BINARY(31) GLOBALREF,            /* Temp hold area for MCA$L_INT_MULT */
933  1      INTERVAL_DEL     BIT(64) ALIGNED  GLOBALREF,            /* Delta time value for Interval */
934  1      SETIMR_STR       FIXED BINARY(7)  GLOBALREF;            /* Count byte for $SETIMR cstring */
935  1
936  1      Declare
937  1      FLUSH_IND        BIT(1) ALIGNED   GLOBALREF,            /* Flush indicator; YES=> perform FLUSH */
938  1      FLUSH_COLLS      FIXED BINARY(15) GLOBALREF,            /* Number of collection events between FLUSH's */
939  1      FLUSH_CTR        FIXED BINARY(15) GLOBALREF;            /* Down counter for FLUSH_COLLS */
940  1
941  1      Declare
942  1      CDBPTR                    POINTER GLOBALREF,            /* Pointer to CDB (Class Descriptor Block) */
943  1      C                         POINTER DEFINED(CDBPTR),      /* Synonym for CDBPTR */
944  1      MRBPTR                    POINTER GLOBALREF,            /* Pointer to MRB (Monitor Request Block) */
945  1      M                         POINTER DEFINED(MRBPTR),      /* Synonym for MRBPTR */
946  1      MCAPTR                    POINTER GLOBALREF,            /* Pointer to MCA (Monitor Communication Area) */
947  1      MC                        POINTER DEFINED(MCAPTR),      /* Synonym for MCAPTR */
948  1      SPTR                      POINTER GLOBALREF,            /* Pointer to SYI (System Information Area) */
949  1      CCDPTR                    POINTER GLOBALREF;            /* Pointer to CCD (Current Class Descriptor) Array */
950  1
951  1      Declare
952  1      INPUT_FILE       FILE RECORD INPUT,                          /* Monitor Input (Playback) File */
953  1      INPUT_CPTR       POINTER GLOBALREF,                          /* Ptr to input buffer count word */
954  1      INPUT_DATA       CHAR(512) VARYING BASED(INPUT_CPTR);        /* Playback file input buffer */
955  1
956  1      Declare
957  1      01 CURR_CLASS_DESCR (MAX_CLASS_NO+1) BASED(CCDPTR),          /* Current Class Descriptor */
958  1                                                                  /* This array of structures includes a CCD (Current
959  1                                                                  /* Class Descriptor) for each possible class     */
960  1          02 CURR_CDBPTR      POINTER,                            /* CDBPTR for current class */
961  1          02 CURR_CLASS_NO    FIXED BINARY(7);                    /* Class number for current class */
962  1
```

```
963  1      /*
964  1      /*      +-----------------------------------------------------------------------+
965  1      /*      |                                                                       |
966  1      /*      |                    GLOBAL STORAGE  DEFINITIONS                        |
967  1      /*      |                                                                       |
968  1      /*      +-----------------------------------------------------------------------+
969  1      /*/
970  1
971  1      /*
972  1      /*      +-----------------------------------------------------------------------+
973  1      /*      |                                                                       |
974  1      /*      |                    COMPILE-TIME CONSTANTS                             |
975  1      /*      |                                                                       |
976  1      /*      +-----------------------------------------------------------------------+
977  1      /*/
978  1
979  1      %REPLACE NOT_SUCCESSFUL         BY '0'B;            /* Failing status bit */
980  1      %REPLACE YES                    BY '1'B;            /* For general use */
981  1      %REPLACE NO                     BY '0'B;            /* For general use */
982  1      /*
983  1      /*      +-----------------------------------------------------------------------+
984  1      /*      |                                                                       |
985  1      /*      |                    OWN STORAGE                                        |
986  1      /*      |                                                                       |
987  1      /*      +-----------------------------------------------------------------------+
988  1      /*/
989  1
990  1      Declare
991  1      CALL                    FIXED BINARY(31),           /* Holds function value (return status) of called ro
992  1      STATUS                  BIT(1)  BASED(ADDR(CALL)),  /* Low-order status bit for called routines */
993  1      I                       FIXED BINARY(15),           /* Index for DO loop */
994  1      BUFF_PTR                POINTER,                    /* Temporary pointer to input file buffer */
995  1      CURR_TYPE               FIXED BINARY(7),            /* Class record type of record just read */
996  1      PREV_TYPE               FIXED BINARY(7),            /* Class record type of record previously read */
997  1      PREV_CONT               BIT(1) ALIGNED,             /* Value of MNR_CLS$V_CONT for record previously rea
998  1      CLASS_MISSING           BIT(1) ALIGNED,             /* For Playback, ON => requested class not in file *
999  1      CLASS_FOUND             BIT(1) ALIGNED;             /* For Playback, ON => requested class found in file
1000 1
1001 1      Declare
1002 1      MON_ERR         ENTRY (ANY VALUE, ANY, ANY) OPTIONS(VARIABLE),  /* MONITOR MACRO-32 routine to log synchronous error
1003 1      READ_INPUT      ENTRY (FIXED BINARY(7)),                        /* MONITOR routine to read an input (playback) file
1004 1      COLLECTION_END  ENTRY,                                          /* MONITOR routine to indicate end of collection */
1005 1      CLASS_COLLECT   ENTRY   (FIXED BINARY(7))                       /* MONITOR MACRO-32 routine to collect a buffer of d
1006 1                              RETURNS(FIXED BINARY(31));
1007 1
```

```
1008  1      /*
1009  1      /*++
1010  1      /*
1011  1      /* FUNCTIONAL DESCRIPTION:
1012  1      /*
1013  1      /*      COLLECTION_EVENT
1014  1      /*
1015  1      /*      COLLECTION_EVENT is an AST routine invoked via the $DCLAST
1016  1      /*      system service from the EXECUTE_REQUEST routine, or via
1017  1      /*      the $SETIMR system service from a previous invocation of
1018  1      /*      COLLECTION_EVENT. It performs performance data collection
1019  1      /*      from VMS data bases of the running system, or from an
1020  1      /*      input recording file. A single invocation of COLLECTION_EVENT
1021  1      /*      causes collection of data for all classes in the MONITOR
1022  1      /*      request. The data is collected by calling the CLASS_COLLECT
1023  1      /*      routine once for each class. CLASS_COLLECT stores the data in a
1024  1      /*      collection buffer (pointed to by the CDB) for each class.
1025  1      /*
1026  1      /*      On the first collection event, class-specific initialization
1027  1      /*      is performed by a call to the CLASS_INIT routine.
1028  1      /*
1029  1      /* INPUTS:
1030  1      /*
1031  1      /*      None
1032  1      /*
1033  1      /* IMPLICIT INPUTS:
1034  1      /*
1035  1      /*      All MONITOR variables accessible to this routine.
1036  1      /*
1037  1      /* OUTPUTS:
1038  1      /*
1039  1      /*      None
1040  1      /*
1041  1      /* IMPLICIT OUTPUTS:
1042  1      /*
1043  1      /*      MCA$L_COLLCNT is incremented by 1.
1044  1      /*
1045  1      /*      Data for all requested classes has been collected into
1046  1      /*      their respective CDB collection buffers.
1047  1      /*
1048  1      /*      All MONITOR variables accessible to this routine.
1049  1      /*
1050  1      /* ROUTINE VALUE:
1051  1      /*
1052  1      /*      COLL_STATUS contains the status of this collection event upon
1053  1      /*      exit.
1054  1      /*
1055  1      /* SIDE EFFECTS:
1056  1      /*
1057  1      /*      If this is the final collection event, the COLLENDED bit is set.
1058  1      /*
1059  1      /*--
1060  1      /*/
1061  1
```

```
1062    1       IF COLLENDED = YES  THEN RETURN;                        /* If collection has already ended, return immediate
1063    1
1064    1       IF M->MRB$V_PLAYBACK                                    /* Playback Request */
1065    1           THEN DO;
1066    2                IF MC->MCA$L_COLLCNT = 0                       /* If first collection event, */
1067    2                    THEN MULT_TEMP = 1;                        /* ... set multiple to trigger on this collection */
1068    2                MC->MCA$V_MULTFND = NO;                        /* Indicate multiple not yet found */
1069    2                MULT_TEMP = MULT_TEMP - 1;                     /* Count down toward zero */
1070    2                IF MULT_TEMP = 0                               /* If it's time to record and display, */
1071    2                    THEN DO;
1072    3                         MC->MCA$V_MULTFND = YES;              /* ... indicate so */
1073    3                         MULT_TEMP = MC->MCA$L_INT_MULT;       /* ... and re-load multiple value for next collectio
1074    3                         MC->MCA$L_CONSEC_REC = MC->MCA$L_CONSEC_REC + 1; /* ... also update to a new consec no for recordi
1075    3                         END;
1076    2                BUFF_PTR = MC->MCA$A_INPUT_PTR;                /* Get pointer to input file buffer for later use */
1077    2                PREV_TYPE = -1;                                /* Dummy previous record type (class no) */
1078    2                PREV_CONT = NO;                                /* Dummy previous "continue" bit setting */
1079    2                CLASS_MISSING = '0'B;                          /* Class not missing */
1080    2
1081    2                DO I = 1 TO M->MRB$W_CLASSCT WHILE (^ MC->MCA$V_EOF & ^ CLASS_MISSING);       /* Loop through all requeste
1082    3                CLASS_FOUND = '0'B;                            /* Haven't found class yet */
1083    3                CDBPTR = CURR_CDBPTR(I);                       /* Set up current CDB */
1084    3                IF MC->MCA$L_COLLCNT = 0                       /* If first collection event */
1085    3                    THEN CALL = CLASS_INIT();                  /* ... then do init for this class */
1086    3
1087    3                DO WHILE (^ MC->MCA$V_EOF & ^ CLASS_FOUND & ^ CLASS_MISSING);  /* Loop causes input file to skip past unwan
1088  ! 4                                                                /* ... classes within the recorded interval
1089    4                CURR_TYPE = BUFF_PTR->MNR_CLS$B_TYPE;          /* Get class (record) type of current record */
1090    4                IF (CURR_TYPE < PREV_TYPE)  !  (CURR_TYPE > CURR_CLASS_NO(I)) !
1091    4                   (CURR_TYPE = PREV_TYPE & PREV_CONT = NO)    /* Check for missing class (should never occur) */
1092    4                    THEN DO;
1093    5                         CLASS_MISSING = YES;                  /* Indicate "class missing" error */
1094    5                         COLL_STATUS = MNR$_CLASMISS;          /* Save failing status */
1095    5                         CALL MON_ERR(MNR$_CLASMISS);          /* ... and log the error */
1096    5                         END;
1097    4                    ELSE DO;
1098    5                         IF CURR_TYPE = CURR_CLASS_NO(I)       /* If inputted class = monitored class */
1099    5                             THEN DO;
1100    6                                  CLASS_FOUND = YES;           /* Indicate found the record needed */
1101    6                                  CALL = CLASS_COLLECT(CURR_CLASS_NO(I)); /* Collect data for this class */
1102    6                                  IF STATUS = NOT_SUCCESSFUL                  /* If collection failed, */
1103    6                                      THEN DO;
1104    7                                           COLL_STATUS = MNR$_COLLERR;        /* Save failing status */
1105    7                                           CALL MON_ERR(MNR$_COLLERR,CALL);   /* Log the error */
1106    7                                           CALL COLLECTION_END();             /* ... and terminate collection */
1107    7                                           END;
1108    6                                  END;
1109    5                         PREV_TYPE = CURR_TYPE;                /* Current now becomes previous */
1110    5                         PREV_CONT = BUFF_PTR->MNR_CLS$V_CONT; /* Save previous "continue" bit setting */
1111    5                         CALL READ_INPUT(SKIP_TO_CLASS);       /* Read the next class record */
1112    5                         END;
1113    4                END;
1114    3
1115    3                IF MC->MCA$V_EOF  !  CLASS_MISSING            /* If anything but CLASS_FOUND, */
1116    3                    THEN CALL COLLECTION_END();                /* ... then indicate collection ended */
1117    3
```

```
1118    3               END;
1119    2
```

```
1120    2               IF COLLENDED = NO                                   /* If end of collection not indicated, then scan */
1121    ;2              /*                                                  /* ... the input file for the beginning of the next
1122    ;2              /*                                                  /* ... interval and leave the file positioned there.
1123    2                   THEN DO;
1124    3                       CURR_TYPE = BUFF_PTR->MNR_CLS$B_TYPE;       /* Get class (record) type of current record */
1125    3                       DO WHILE(^ MC->MCA$V_EOF & CURR_TYPE > PREV_TYPE); /* Loop while class type numbers increase */
1126    4                       PREV_TYPE = CURR_TYPE;                      /* Current now becomes previous */
1127    4                       CALL READ_INPUT(SKIP_TO_CLASS);             /* Read the next class record */
1128    4                       CURR_TYPE = BUFF_PTR->MNR_CLS$B_TYPE;       /* Get class (record) type of current record */
1129    4                       END;
1130    3                       IF MC->MCA$V_EOF                            /* If end-of-file reached, */
1131    3                           THEN CALL COLLECTION_END();             /* ... then indicate so */
1132    3                       END;
1133    2
1134    2               IF MC->MCA$V_EOF  &  MC->MCA$L_COLLCNT = 0          /* If end-of-file after first collection event, */
1135    2                   THEN DO;                                        /* ... then this is an error */
1136    3                       COLL_STATUS = MNR$_BEGNLEND;                /* Save failing status */
1137    3                       CALL MON_ERR(MNR$_BEGNLEND);                /* ... and log the error */
1138    3                       END;
1139    2
1140    2               END;                                                /* End of Playback Request processing */
1141    1
1142    1
```

```
1143    1           ELSE DO;                                                    /* Live Request */
1144    2
1145    2                   MC->MCA$L_CONSEC_REC = MC->MCA$L_CONSEC_REC + 1;    /* Update to a new consec no for recording */
1146    2                   IF M->MRB$V_RECORD                                  /* If recording, */
1147    2                       THEN FLUSH_CTR = FLUSH_CTR - 1;                 /* Decrement flush counter for this coll event */
1148    2
1149    2                   DO I = 1 TO M->MRB$W_CLASSCT WHILE (COLLENDED = NO); /* Loop once for each requested class */
1150    3                   CDBPTR = CURR_CDBPTR(I);                            /* Set up current CDB */
1151    3                   IF MC->MCA$L_COLLCNT = 0                            /* If first collection event */
1152    3                       THEN CALL = CLASS_INIT();                       /* ... then do init for this class */
1153    3
1154    3                   IF FLUSH_CTR = 0  &  CURR_CLASS_NO(I) = MC->MCA$B_LASTC /* If FLUSH_CTR reached zero, and this is last cl
1155    3                       THEN DO;                                        /* ... then time to flush */
1156    4                            FLUSH_IND = YES;                           /* Indicate flush required */
1157    4                            FLUSH_CTR = FLUSH_COLLS;                   /* ... and start down counter at beginning again */
1158    4                            END;
1159    3
1160    3                   CALL = CLASS_COLLECT(CURR_CLASS_NO(I));             /* Collect data for this class */
1161    3                   IF STATUS = NOT_SUCCESSFUL                          /* If collection failed, */
1162    3                       THEN DO;
1163    4                            COLL_STATUS = MNR$_COLLERR;                /* Save failing status */
1164    4                            CALL MON_ERR(MNR$_COLLERR,CALL);           /* Log the error */
1165    4                            CALL COLLECTION_END();                     /* ... and terminate collection */
1166    4                            END;
1167    3                   END;
1168    2
1169    2                   IF COLLENDED = NO                                   /* If not at end of collection, */
1170    2                       THEN DO;
1171    3                            CALL = SYS$SETIMR(COLL_EV_FLAG,INTERVAL_DEL,COLLECTION_EVENT,);
1172 ;  3                                                                       /* Re-enter COLLECTION_EVENT at specified interval *
1173 ;  3                                                                       /* COLL_EV_FLAG is not used; it is just a dummy */
1174    3                            IF STATUS = NOT_SUCCESSFUL                 /* If $SETIMR failed, */
1175    3                                THEN DO;
1176    4                                     COLL_STATUS = MNR$_SSERROR;       /* Save failing status */
1177    4                                     CALL MON_ERR(MNR$_SSERROR,CALL,SETIMR_STR); /* Log the error */
1178    4                                     CALL COLLECTION_END();            /* ... and terminate collection */
1179    4                                     END;
1180    3                            END;
1181    2                   END;                                                /* End of Live Request processing */
1182    1
1183    1           MC->MCA$L_COLLCNT = MC->MCA$L_COLLCNT + 1;                  /* Count this collection event */
1184    1           RETURN;                                                     /* Return to caller */
1185    1
```

```
1186    1       CLASS_INIT: Procedure Returns(fixed binary(31));                /* Class-specific initialization */
1187    2
1188    2       /*
1189    2       /*++
1190    2       /*
1191    2       /* FUNCTIONAL DESCRIPTION:
1192    2       /*
1193    2       /*      CLASS_INIT
1194    2       /*
1195    2       /*      This routine is called by COLLECTION_EVENT on the first
1196    2       /*      collection event to perform class-specific initialization.
1197    2       /*      Currently, the MODES, PROCESSES, DISK, DLOCK and SYSTEM
1198    2       /*      classes require such initialization.
1199    2       /*
1200    2       /* INPUTS:
1201    2       /*
1202    2       /*      None
1203    2       /*
1204    2       /* OUTPUTS:
1205    2       /*
1206    2       /*      None
1207    2       /*
1208    2       /* IMPLICIT OUTPUTS:
1209    2       /*
1210    2       /*      Initialization for the MODES, PROCESSES, DISK, DLOCK and SYSTEM classes performed.
1211    2       /*
1212    2       /* ROUTINE VALUE:
1213    2       /*
1214    2       /*      SS$_NORMAL
1215    2       /*
1216    2       /* SIDE EFFECTS:
1217    2       /*
1218    2       /*      None
1219    2       /*
1220    2       /*--
1221    2       /*/
1222    2
```

```
1223  |  2        /*
1224  |  2        /*      |-------------------------------------------------------------------------+
1225  |  2        /*      |                                                                         |
1226  |  2        /*      |                              LOCAL STORAGE                              |
1227  |  2        /*      |                                                                         |
1228  |  2        /*      |-------------------------------------------------------------------------+
1229  |  2        /*/
1230     2
1231     2        Declare
1232     2        PROCS_CLSNO     FIXED BINARY(31) GLOBALREF VALUE,          /* PROCESSES class number */
1233     2        MODES_CLSNO     FIXED BINARY(31) GLOBALREF VALUE,          /* MODES class number */
1234     2        DISK_CLSNO      FIXED BINARY(31) GLOBALREF VALUE,          /* DISK class number */
1235     2        DLOCK_CLSNO     FIXED BINARY(31) GLOBALREF VALUE,          /* DLOCK class number */
1236     2        SYSTEM_CLSNO    FIXED BINARY(31) GLOBALREF VALUE,          /* SYSTEM class number */
1237     2        TOP_RANGE       FIXED BINARY(31) GLOBALREF VALUE,          /* Range value for TOPB, TOPD, TOPF bar graph */
1238     2        MODES_STRLEN    FIXED BINARY(31) GLOBALREF VALUE;          /* Length of "Interrupt Stack" string */
1239     2
1240     2        Declare
1241     2        IDBPTR          POINTER,                                   /* Pointer to Item Descriptor Block (IDB) */
1242     2        ITMSTR (1:C->CDB$L_ICOUNT) BIT(8) ALIGNED BASED(C->CDB$A_ITMSTR), /* Vector of item numbers for this class */
1243     2        ITEM_IDX        FIXED BINARY(15),                          /* Index into IDB_BLOCK */
1244     2        ITEMNO          FIXED BINARY (7);                          /* Item number used in DO loop */
1245     2
1246     2        Declare
1247     2        1 PERFTABLE GLOBALREF,                                     /* Table of IDB's */
1248     2          2 IDB_BLOCK (0:255) CHAR(IDB$K_ILENGTH);                 /* Up to 256 IDB's */
1249     2
1250     2        Declare
1251     2        1 PINTERRUPT_STR  BASED(IDBPTR->IDB$A_LNAME),              /* Counted string for "Interrupt Stack PRIMARY" */
1252     2          2 L FIXED BINARY(7),                                     /* Count */
1253     2          2 S CHAR(1);                                             /* First character of string */
1254     2
1255     2        Declare
1256     2        REVLEVELS       (0:127) FIXED BINARY(7) GLOBALREF;         /* Revision Levels Vector */
1257     2
1258     2        Declare
1259     2        1 DIR_STR  BASED(IDBPTR->IDB$A_LNAME),                     /* Counted string for "Directory" items in DLOCK */
1260     2          2 L FIXED BINARY(7),                                     /* Count */
1261     2          2 X CHAR(17),                                            /* Uninteresting characters of string */
1262     2          2 S CHAR(5);                                             /* Characters of interest */
1263     2
1264     2        Declare
1265     2        PROCTITLE (0:127) GLOBALREF POINTER;                       /* Table of pointers to PROCESSES screen titles */
1266     2
1267     2        Declare
1268     2        1 BU_SYS_SINGLE         GLOBALREF,                         /* Bar graph range values for SYSTEM class (single s
1269     2          2 BSS_RANGE (1:17)    FIXED BINARY(31);
1270     2
```

```
1271     2      IF CURR_CLASS_NO(I) = MODES_CLSNO                                  /* If MODES class, */
1272     2          THEN DO;
1273     3               C->CDB$V_CPU_COMB = NO;                                   /* Assume no CPU combination necessary */
1274     3               MC->MCA$A_MPADDR = NULL();                                /* Indicate no MP address */
1275     3               UNSPEC(ITEM_IDX) = ITMSTR(1);                             /* Zero-extend ITMSTR element to word */
1276     3               IDBPTR = ADDR(IDB_BLOCK(ITEM_IDX));                       /* Set up IDB ptr in order to */
1277     3     !                                                                   /* ... reference PINTERRUPT_STR */
1278     3               PINTERRUPT_STR.L = MODES_STRLEN;                          /* Get length of "Interrupt Stack" string */
1279     3               IF SPTR->MNR_SYI$B_MPCPUS = 2                             /* Check if monitored system a multiprocesso
1280     3                   THEN DO;                                              /* Multiprocessor system */
1281     4                        C->CDB$L_ICOUNT = C->CDB$L_ICOUNT *              /* Increase number of collected items */
1282     4                                          SPTR->MNR_SYI$B_MPCPUS;
1283     4                        C->CDB$W_BLKLEN = C->CDB$W_BLKLEN *              /* ... and size of coll buff data block */
1284     4                                          SPTR->MNR_SYI$B_MPCPUS;
1285     4                        IF C->CDB$V_CPU & M->MRB$V_SYSCLS = NO & M->MRB$V_MFSUM = NO /* If CPU-specific display requested
1286     4     !                                                                  /* AND SYSTEM class not being monitored, */
1287     4                            THEN DO;                                    /* AND not multi-file summary, */
1288     5                                 C->CDB$L_ECOUNT = C->CDB$L_ICOUNT;     /* Increase number of displayed elements */
1289     5                                 END;
1290     4                            ELSE DO;                                    /* Combined display */
1291     5                                 C->CDB$V_CPU_COMB = YES;               /* Indicate that collected items must be */
1292     5     !                                                                  /* ... combined for display */
1293     5                                 PINTERRUPT_STR.L = PINTERRUPT_STR.L - 10; /* Shorten "Interrupt Stack" display string
1294     5     !                                                                  /* ... to remove the word "PRIMARY" */
1295     5                                 END;
1296     4                        END;
1297     4                    ELSE                                                /* Uniprocessor system */
1298     4                        PINTERRUPT_STR.L = PINTERRUPT_STR.L - 10;       /* Shorten "Interrupt Stack" display string
1299     4     !                                                                  /* ... to remove the word "PRIMARY" */
1300     3               END;
1301     3
1302     2      IF CURR_CLASS_NO(I) = PROCS_CLSNO                                  /* If PROCESSES class, */
1303     2          THEN DO;
1304     3               C->CDB$A_TITLE = PROCTITLE(C->CDB$B_ST);                  /* Set up ptr to title for requested display
1305     3
1306     3               IF C->CDB$B_ST = TOPC_PROC                                /* If TOPCPU display, */
1307     3                   THEN C->CDB$L_RANGE = 100;                            /* Set range to 100 */
1308     3                   ELSE C->CDB$L_RANGE = TOP_RANGE;                      /* Set range for other TOP displays */
1309     3
1310     3               END;
1311     2
```

```
1312    2       IF CURR_CLASS_NO(I) = DISK_CLSNO                        /* If DISK class ... */
1313    2        & REVLEVELS(DISK_CLSNO) > 0                            /* ... AND it is any rev level after 0, */
1314    2           THEN DO;
1315    3                C->CDB$V_DISKAC = YES;                         /*   then indicate DISK with allocation clas
1316    3                IF REVLEVELS(DISK_CLSNO) > 1                   /*   If any rev level after 1, */
1317    3                    THEN C->CDB$V_DISKVN = YES;                /*     then indicate DISK with volume names
1318    3                    ELSE C->CDB$V_DISKVN = NO;                 /*     else indicate not */
1319    3                END;
1320    2           ELSE DO;
1321    3                C->CDB$V_DISKAC = NO;                          /*   else indicate no alloc class in name, *
1322    3                C->CDB$V_DISKVN = NO;                          /*   ... and no volume name */
1323    3                END;
1324    2
1325    2       IF CURR_CLASS_NO(I) = DISK_CLSNO                        /* If DISK class ... */
1326    2        & M->MRB$V_MFSUM = NO                                  /* ... AND it's not m.f. summary, */
1327    2        & C->CDB$B_ST = ALL_STAT                               /* ... AND all stats requested, */
1328    2           THEN C->CDB$V_WIDE = YES;                           /*   then indicate wide display, */
1329    2           ELSE C->CDB$V_WIDE = NO;                            /*   else indicate usual width */
1330    2
1331    2       IF CURR_CLASS_NO(I) = DLOCK_CLSNO                       /* If DLOCK class ... */
1332    2        & REVLEVELS(DLOCK_CLSNO) = 0                           /* ... AND it is rev level 0, */
1333    2           THEN DO;
1334    3
1335    3                DO ITEMNO = 13 TO 15 BY 1;                     /* Change text for last three items */
1336    4
1337    4                UNSPEC(ITEM_IDX) = ITMSTR(ITEMNO);             /* Zero-extend ITMSTR element to word */
1338    4                IDBPTR = ADDR(IDB_BLOCK(ITEM_IDX));            /* Set up IDB ptr in order to */
1339  ; 4                                                             /* ... reference DIR_STR */
1340    4                IF SPTR->MNR_SYI$V_RESERVED1                   /* If this is directory node, */
1341    4                    THEN DIR_STR.S = 'Incom';                  /*   then rates are "Incoming" */
1342    4                    ELSE DIR_STR.S = 'Outgo';                  /*   else they are "Outgoing" */
1343    4
1344    4                END;                                           /* End of DO loop */
1345    3
1346    3                END;
1347    2
1348    2
```

```
1349    2       IF CURR_CLASS_NO(I) = SYSTEM_CLSNO                              /* If SYSTEM class, */
1350    2           THEN DO;
1351    3                BSS_RANGE(14) = SPTR->MNR_SYI$L_BALSETMEM;             /* Stash away range of Free List bar graph *
1352    3                BSS_RANGE(15) = SPTR->MNR_SYI$L_MPWHILIM;              /* ... and Modified List bar graph */
1353    3                END;
1354    2
1355    2       RETURN(NORMAL);
1356    2
1357    2       END CLASS_INIT;
1358    1
1359    1       END COLLECTION_EVENT;
```

COMMAND LINE

```
PLI/LIS=LIS$:COLLEVT/OBJ=OBJ$:COLLEVT MSRC$:COLLEVT+LIB$:MONLIB/LIB
```

0239 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

COLLEVT
LIS

MOMTEST
LIS

MONDEF
SDL

GETBUFF
LIS

MONSUB
CLD

NPARSE
LIS

MONTOR

MONITOR
MAP

DSPDEF
MDL