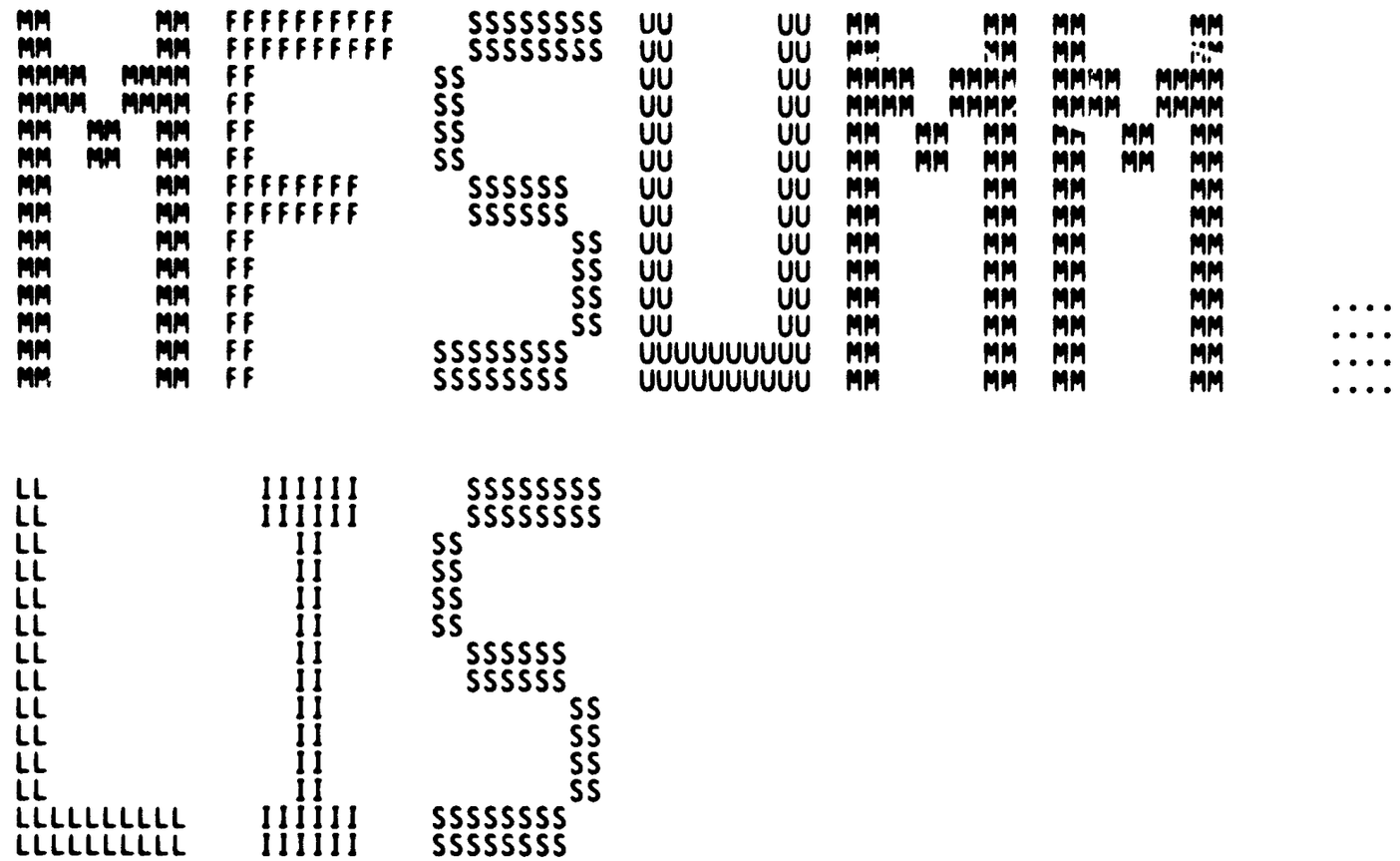
```
MMM           MMM     000000000     NNN           NNN  TTTTTTTTTTTTTTT     000000000     RRRRRRRRRRR
MMM           MMM     000000000     NNN           NNN  TTTTTTTTTTTTTTT     000000000     RRRRRRRRRRRR
MMM           MMM     000000000     NNN           NNN  TTTTTTTTTTTTTTT     000000000     RRRRRRRRRRRR
MMMMMM     MMMMMM  000           000  NNN           NNN        TTT        000           000  RRR         RRR
MMMMMM     MMMMMM  000           000  NNN           NNN        TTT        000           000  RRR         RRR
MMMMMM     MMMMMM  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
```


MM MM FFFFFFFFFF SSSSSSSS UU UU MM MM MM MM
MM MM FFFFFFFFFF SSSSSSSS UU UU MM MM MM MM
MMMM MMMM FF SS UU UU MMMM MMMM MMMM MMMM
MMMM MMMM FF SS UU UU MMMM MMMM MMMM MMMM
MM MM MM FF SS UU UU MM MM MM MM MM MM
MM MM MM FF SS UU UU MM MM MM MM MM MM
MM MM FFFFFFF SSSSSS UU UU MM MM MM MM
MM MM FFFFFFF SSSSSS UU UU MM MM MM MM
MM MM FF SS UU UU MM MM MM MM
MM MM FF SS UU UU MM MM MM MM
MM MM FF SS UU UU MM MM MM MM ....
MM MM FF SS UU UU MM MM MM MM ....
MM MM FF SSSSSSSS UUUUUUUUUU MM MM MM MM ....
MM MM FF SSSSSSSS UUUUUUUUUU 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
LLLLLLLLLL IIIIII SSSSSSSS
LLLLLLLLLL IIIIII SSSSSSSS

```
    1           MFSUM_REQUEST: Procedure        Returns(Fixed Binary(31))                   /* Routine to execute multi-file summary req
    2                                           Options(Ident('V04-000'));
    3     1
    4     1     /*
    5     1     /*****************************************************************************
    6     1     /*                                                                           *
    7     1     /*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                  *
    8     1     /*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                   *
    9     1     /*  ALL RIGHTS RESERVED.                                                     *
   10     1     /*                                                                           *
   11     1     /*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED    *
   12     1     /*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE   *
   13     1     /*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER    *
   14     1     /*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY    *
   15     1     /*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY    *
   16     1     /*  TRANSFERRED.                                                             *
   17     1     /*                                                                           *
   18     1     /*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE    *
   19     1     /*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT    *
   20     1     /*  CORPORATION.                                                             *
   21     1     /*                                                                           *
   22     1     /*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS    *
   23     1     /*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                  *
   24     1     /*                                                                           *
   25     1     /*                                                                           *
   26     1     /*****************************************************************************
   27     1     /*/
   28     1
   29     1     /*
   30     1     /*++
   31     1     /* FACILITY:  MONITOR Utility
   32     1     /*
   33     1     /* ABSTRACT:  MFSUM_REQUEST Routine.
   34     1     /*
   35     1     /*               Called from MONMAIN routine to execute a single
   36     1     /*                 MONITOR multi-file summary request.
   37     1     /*
   38     1     /*
   39     1     /* ENVIRONMENT:
   40     1     /*
   41     1     /*              Unprivileged user mode.
   42     1     /*
   43     1     /* AUTHOR:  Thomas L. Cafarella, January, 1984
   44     1     /*
   45     1     /*
   46     1     /* MODIFIED BY:
   47     1     /*
   48     1     /*      V03-006 TLC1088         Thomas L. Cafarella     25-Jul-1984     14:00
   49     1     /*              Free virtual memory obtained for multi-file summary.
   50     1     /*
   51     1     /*      V03-005 TLC1077         Thomas L. Cafarella     11-Jul-1984     14:00
   52     1     /*              Continue multi-file summary even if some input files are empty.
   53     1     /*
   54     1     /*      V03-004 TLC1071         Thomas L. Cafarella     17-Apr-1984     14:00
   55     1     /*              Add "number of input files" to multi-file summary report.
```

```
    56  :   1       /*
    57  :   1       /*      V03-003 TLC1069         Thomas L. Cafarella     13-Apr-1984     14:00
    58  :   1       /*              Fix ACCVIO when multi-file summary requested for SYSTEM class
    59  :   1       /*              and single-statistic.
    60  :   1       /*
    61  :   1       /*      V03-002 PRS1021         Paul R. Senn            12-Apr-1984     10:00
    62  :   1       /*              Make SYSTEM class work with multi-file summary.
    63  :   1       /*
    64  :   1       /*      V03-002 PRS1020         Paul R. Senn            12-Apr-1984     10:00
    65  :   1       /*              Make XQP class work with multi-file summary.
    66  :   1       /*
    67  :   1       /*      V03-002 TLC1060         Thomas L. Cafarella     12-Mar-1984     11:00
    68  :   1       /*              Make multi-file summary work for homogeneous classes.
    69  :   1       /*
    70  :   1       /*      V03-001 TLC1053         Thomas L. Cafarella     07-Mar-1984     11:00
    71  :   1       /*              Add support to ignore unused files for multi-file summary.
    72  :   1       /*
    73  :   1       /*--
    74  :   1       /*/
    75      1
```

```
    1  1        /*++
    2  1        /*
    3  1        /* FUNCTIONAL DESCRIPTION:
    4  1        /*
    5  1        /*      MFSUM_REQUEST
    6  1        /*
    7  1        /*      TBS
    8  1        /*
    9  1        /* INPUTS·
   10  1        /*
   11  1        /*      None
   12  1        /*
   13  1        /* IMPLICIT INPUTS:
   14  1        /*
   15  1        /*      Monitor Request Block (MRB), pointed to by MRBPTR.
   16  1        /*
   17  1        /* OUTPUTS:
   18  1        /*
   19  1        /*      None
   20  1        /*
   21  1        /* IMPLICIT OUTPUTS:
   22  1        /*
   23  1        /*      TBS
   24  1        /*
   25  1        /* ROUTINE VALUE:
   26  1        /*
   27  1        /*      SS$_NORMAL, or failing MONITOR status code.
   28  1        /*
   29  1        /* SIDE EFFECTS:
   30  1        /*
   31  1        /*      None
   32  1        /*
   33  1        /*/
   34  1
```

```
110  1      /*
111  1      /*      !-------------------------------------------------------------------------+
112  1      /*      !                                                                         !
113  1      /*      !                              INCLUDE  FILES                             !
114  1      /*      !                                                                         !
115  1      /*      !-------------------------------------------------------------------------+
116  1      /*/
117  1
118  1      %INCLUDE        MONDEF;                                 /* Monitor utility structure definitions */
886  1      %INCLUDE        $CHFDEF;                                /* Condition handler facility definitions */
906  1      %INCLUDE        $STSDEF;                                /* Status value definitions */
923  1      %INCLUDE        SYS$CANCEL;                             /* $CANCEL system service */
929  1      %INCLUDE        SYS$PUTMSG;                             /* $PUTMSG system service */
937  1
938  1
939  1      /*
940  1      /*      !-------------------------------------------------------------------------+
941  1      /*      !                                                                         !
942  1      /*      !                    EXTERNAL STORAGE  DEFINITIONS                        !
943  1      /*      !                                                                         !
944  1      /*      !-------------------------------------------------------------------------+
945  1      /*/
946  1
947  1      Declare
948  1      MAX_CLASS_NO    FIXED BINARY(31) GLOBALREF VALUE,       /* Maximum defined class number */
949  1      PROCS_CLSNO     FIXED BINARY(31) GLOBALREF VALUE,       /* Class number for the PROCESSES class */
950  1      MAXELTS_MFS     FIXED BINARY(31) GLOBALREF VALUE,       /* Max no. of elements for a homogeneous class m.f.
951  1      MAX_ELIDLEN     FIXED BINARY(31) GLOBALREF VALUE,       /* Max length of an element id for a homogeneous cla
952  1      MAX_HOM_ITEMS   FIXED BINARY(31) GLOBALREF VALUE,       /* Max no. of items for a homogeneous class */
953  1      MAX_INP_FILES   FIXED BINARY(31) GLOBALREF VALUE,       /* Max no. of input files for multi-file summary */
954  1      ECOUNT_SYS_ALL  FIXED BINARY(31) GLOBALREF VALUE;       /* Element count for SYS/ALL */
955  1
956  1      Declare
957  1      CDBPTR                  POINTER GLOBALREF,              /* Pointer to CDB (Class Descriptor Block) */
958  1      C                       POINTER DEFINED(CDBPTR),        /* Synonym for CDBPTR */
959  1      MCAPTR                  POINTER GLOBALREF,              /* Pointer to MCA (Monitor Communication Area) */
960  1      MC                      POINTER DEFINED(MCAPTR),        /* Synonym for MCAPTR */
961  1      MRBPTR                  POINTER GLOBALREF,              /* Pointer to MRB (Monitor Request Block) */
962  1      M                       POINTER DEFINED(MRBPTR);        /* Synonym for MRBPTR */
963  1
964  1      Declare
965  1      H                       POINTER GLOBALREF;              /* Pointer to input file header */
966  1
967  1      Declare
968  1      INPUT_FILE      FILE RECORD INPUT;                      /* Monitor Input (Playback) File */
969  1
970  1      Declare
971  1      CTRLCZ_CHAN     FIXED BINARY(31) GLOBALREF,             /* Channel number for CTRL-C and CTRL-Z */
972  1      CTRLCZ_HIT      BIT(1) ALIGNED   GLOBALREF,             /* YES => CTRL-C or CTRL-Z has been hit */
973  1      NORMAL          FIXED BINARY(31) GLOBALREF;             /* MONITOR normal return status */
974  1
975  1      Declare
976  1      1 CDBHEAD GLOBALREF,                                    /* Table of CDB's */
977  1        2 CDBLOCK (0:127) CHAR(CDB$K_SIZE);
978  1
979  1
```

```
/*
/*      !-----------------------------------------------------------------------+
/*      !                                                                       !
/*      !                        EXTERNAL ROUTINE DEFINITIONS                   !
/*      !                                                                       !
/*      !-----------------------------------------------------------------------+
/*!

Declare
EXECUTE_REQUEST ENTRY RETURNS(FIXED BINARY(31)),                    /* MONITOR routine to execute a MONITOR request */
ESTAB_CTRLCZ    ENTRY RETURNS(FIXED BINARY(31)),                    /* MONITOR MACRO-32 routine to set up CTRL-C and CTR
INPUT_INIT      ENTRY RETURNS(FIXED BINARY(31)),                    /* MONITOR routine to do initialization on input fil
INPUT_CLEANUP   ENTRY RETURNS(FIXED BINARY(31)),                    /* MONITOR routine to do cleanup on input file */
SUMMARY_INIT    ENTRY RETURNS(FIXED BINARY(31)),                    /* MONITOR MACRO-32 routine to do summary init */
SUMMARY_CLEANUP ENTRY RETURNS(FIXED BINARY(31)),                    /* MONITOR routine to do cleanup on summary file */
MON_ERR         ENTRY (ANY VALUE, ANY, ANY) OPTIONS(VARIABLE),      /* MONITOR MACRO-32 routine to log synchronous error
SIGNAL_MON_ERR  ENTRY,                                              /* MONITOR MACRO-32 routine to signal MONITOR errors
LINK_MON_ERR    ENTRY (ANY VALUE, ANY VALUE),                       /* MONITOR MACRO-32 routine to link a MONITOR error
QUAD_LT_QUAD    ENTRY (BIT(64) ALIGNED, BIT(64) ALIGNED)            /* MONITOR MACRO-32 unsigned quadword compare routin
                RETURNS(BIT(1));



/*
/*      !-----------------------------------------------------------------------+
/*      !                                                                       !
/*      !                        MESSAGE DEFINITIONS                            !
/*      !                                                                       !
/*      !-----------------------------------------------------------------------+
/*!

Declare
MNR$_NOCLASS    FIXED BINARY(31) GLOBALREF VALUE,
MNR$_CLASDISAB  FIXED BINARY(31) GLOBALREF VALUE,
MNR$_NOCOMMSTLEV FIXED BINARY(31) GLOBALREF VALUE,
MNR$_PREMEOF    FIXED BINARY(31) GLOBALREF VALUE,
MNR$_INVINPFIL  FIXED BINARY(31) GLOBALREF VALUE,
MNR$_NOCLASSES  FIXED BINARY(31) GLOBALREF VALUE,
MNR$_NOINPFILES FIXED BINARY(31) GLOBALREF VALUE,
MNR$_IGNFIL     FIXED BINARY(31) GLOBALREF VALUE,
MNR$_NOSUMM     FIXED BINARY(31) GLOBALREF VALUE,
MNR$_DISPERR    FIXED BINARY(31) GLOBALREF VALUE;
```

```
: 1       /*
: 1       /*      !---------------------------------------------------------------+
: 1       /*      !                                                               !
: 1       /*      !                 GLOBAL STORAGE  DEFINITIONS                   !
: 1       /*      !                                                               !
: 1       /*      !---------------------------------------------------------------+
: 1       /**/
  1
  1       Declare
  1       MFSPTR            POINTER GLOBALDEF,                      /* Pointer to Multi-File Summary Block (MFS) */
  1       CSBVEC_PTR        POINTER GLOBALDEF,                      /* Pointer to CSB Vector */
  1       1 CSBVEC,                                                 /* CSB Vector (of ptrs to CSBs) */
  1         2 CSB_POINTER (1:MAX_INP_FILES) POINTER;               /* CSB pointers */
  1
: 1       /*
: 1       /*      !---------------------------------------------------------------+
: 1       /*      !                                                               !
: 1       /*      !                   COMPILE-TIME CONSTANTS                      !
: 1       /*      !                                                               !
: 1       /*      !---------------------------------------------------------------+
: 1       /**/
  1
  1       %REPLACE          NOT_SUCCESSFUL          BY '0'B;          /* Failing status bit */
  1       %REPLACE          YES                     BY '1'B;          /* For general use */
  1       %REPLACE          NO                      BY '0'B;          /* For general use */
  1       %REPLACE          AND_OP                  BY '0001'B;       /* AND Boolean operation */
  1       %REPLACE          OR_OP                   BY '0111'B;       /* OR Boolean operation */
  1
```

```
1052  1     /*
1053  1     /*      +-------------------------------------------------------------------+
1054  1     /*      |                                                                   |
1055  1     /*      |                          OWN STORAGE                              |
1056  1     /*      |                                                                   |
1057  1     /*      +-------------------------------------------------------------------+
1058  1     /*/
1059  1
1060  1     Declare
1061  1     CALL            FIXED BINARY(31),                   /* Holds function value (return status) of called ro
1062  1     STATUS          BIT(1)  BASED(ADDR(CALL));          /* Low-order status bit for called routines */
1063  1
1064  1     Declare
1065  1     IFBPTR          POINTER,                            /* Pointer to Input File Block (IFB) */
1066  1     CSBPTR          POINTER,                            /* Pointer to Column Summary Block (CSB) */
1067  1     DCDB            POINTER STATIC;                     /* CDB for current class */
1068  1
1069  1     Declare
1070  1     IFB_TAB_PTR     POINTER,                            /* Pointer to IFB_TABLE */
1071  1     1 IFB_TABLE     BASED(IFB_TAB_PTR),                 /* Input File Block (IFB) Table */
1072  1       2 AN_IFB      (1:MAX_INP_FILES)  CHAR(IFB$K_SIZE); /* A single IFB */
1073  1
1074  1     Declare
1075  1     01 CURR_CLASS_DESCR (MAX_CLASS_NO+1),               /* Current Class Descriptor */
1076  1                                                         /* This array of structures includes a CCD (Current
1077  1                                                         /* Class Descriptor) for each possible class. */
1078  1         02 CURR_CDBPTR      POINTER,                    /* CDBPTR for current class */
1079  1         02 CURR_CLASS_NO    FIXED BINARY(7);            /* Class number for current class */
1080  1
1081  1     Declare
1082  1     1 BUFFERS       BASED(MFS$A_STATSBUF),              /* Statistics buffers */
1083  1       2 TOT (1:MAXELTS_MFS) FIXED BINARY(31),           /* TOTAL statistics buffer */
1084  1       2 MIN (1:MAXELTS_MFS) FIXED BINARY(31),           /* MINIMUM statistics buffer */
1085  1       2 MAX (1:MAXELTS_MFS) FIXED BINARY(31);           /* MAXIMUM statistics buffer */
1086  1
1087  1     Declare
1088  1     I                 FIXED BINARY(15),                 /* Loop control */
1089  1     J                 FIXED BINARY(15),                 /* Loop control */
1090  1     NON_EMPTY_FILES   FIXED BINARY(15),                 /* Count of non-empty input files */
1091  1     ST_REV_COMM       CHAR(8),                          /* Common file structure level */
1092  1     FILE_CLASSES      BIT(128) ALIGNED,                 /* Class bit string from current input file */
1093  1     ALL_FILE_CLASSES  BIT(128) ALIGNED,                 /* Class bit string which is union (OR) of all input
1094  1     CURR_NODENAME     CHAR(16),                         /* Nodename from current input file */
1095  1     FOUND_NODE        BIT(1) ALIGNED,                   /* YES => a matching node was found in a CSB */
1096  1     CURR_ERRCODE      FIXED BINARY(31),                 /* MONITOR error status code currently expected */
1097  1     MF_REQ_STATUS     FIXED BINARY(31),                 /* MFSUM_REQUEST status code */
1098  1     ALREADY_FAILED    BIT(1) ALIGNED;                   /* YES => a failure has already been signaled */
1099  1
```

```
1100    1       ON FINISH;                                              /* On finish, do nothing */
1101    1       ALREADY_FAILED = NO;                                    /* Indicate no failure yet signaled */
1102    1       CURR_ERRCODE = 0;                                       /* Set expected MONITOR code to default */
1103    1
1104    1       /*
1105    1       /*      Set up condition handler to terminate the MONITOR request on:
1106    1       /*              1) any asynchronous error condition, such as file and I/O errors;
1107    1       /*              2) any synchronous MONITOR-detected condition.
1108    1       /*/
1109    1
1110    1       ON ANYCONDITION                                         /* On any condition signaled, */
1111    1       BEGIN;
1112    2
1113    2       Declare
1114    2         MNR$_ERRINPFIL        FIXED BINARY(31) GLOBALREF VALUE,       /* Error message code */
1115    2         MNR$_ERRRECFIL        FIXED BINARY(31) GLOBALREF VALUE,       /* Error message code */
1116    2         MNR$_UNEXPERR         FIXED BINARY(31) GLOBALREF VALUE,       /* Error message code */
1117    2         MON_CODE              FIXED BINARY(31),                       /* Monitor message code */
1118    2         TEMP                  FIXED BINARY(31),                       /* Temporary scratch area */
1119    2         MNR$_FACNO            FIXED BINARY(31) GLOBALREF VALUE,       /* MONITOR facility number */
1120    2         ON_FILE               CHAR(100) VARYING,                      /* Holds possible file name string */
1121    2         SIGNALED_ERR ENTRY (ANY VALUE, ANY VALUE, ANY VALUE, ANY);    /* Rtn to set up PUTMSGVEC */
1122    2
1123    2       IF ^ ALREADY_FAILED                                     /* If a failure not already signaled, */
1124    2           THEN DO;
1125    3                ALREADY_FAILED = YES;                          /* Indicate a failure has been signaled */
1126    3                CHF$ARGPTR = ONARGSLIST();                     /* Get signal array pointer */
1127    3                STS$VALUE = CHF$SIG_NAME;                      /* Get code for signaled condition */
1128    3                UNSPEC(TEMP) = STS$FAC_NO;                     /* Convert facility no. to binary in TEMP */
1129    3                IF TEMP = MNR$_FACNO                           /* If a MONITOR code, */
1130    3                    THEN MON_CODE = STS$VALUE;                 /*      then remember it */
1131    3                    ELSE DO;                                   /* Otherwise, need to set the MON_CODE */
1132    4                         ON_FILE = ONFILE();                   /* Get PL/I file constant if I/O cond */
1133    4                         IF ON_FILE = 'INPUT_FILE'             /* If input file error, */
1134    4                             THEN MON_CODE = MNR$_ERRINPFIL;   /* Set Monitor status code accordingly */
1135    4                             ELSE IF CURR_ERRCODE = 0          /* Else, see if an error is currently expected */
1136    4                                  THEN MON_CODE = MNR$_UNEXPERR;   /* No, set "unexpected" code */
1137    4                                  ELSE MON_CODE = CURR_ERRCODE;    /* Yes, set currently expected code */
1138    4                         CURR_ERRCODE = 0;                     /* Reset to default MONITOR error code ("unexpected"
1139    4                         CALL SIGNALED_ERR(MON_CODE,STS$VALUE,DIM(CHF$SIG_ARG,1),CHF$SIG_ARG); /* Log the error */
1140    4                         END;
1141    3
1142    3                MF_REQ_STATUS = MON_CODE;                      /* Set up code for MONITOR request termination */
1143    3                CALL MF_REQ_CLEANUP();                         /* Perform cleanup for files, memory, etc. */
1144    3                END;
1145    2
1146    2       GO TO MF_REQ_EXIT;                                      /* Go return from MFSUM_REQUEST (PL/I does an UNWIND
1147    2
1148    2       END;                                                    /* End of ON-condition routine */
1149    1
```

```
1150    1       MC->MCA$V_EOF = NO;                                     /* End-of-file not hit yet */
1151    1       ON ENDFILE(INPUT_FILE) MC->MCA$V_EOF = YES;             /* Set up EOF condition */
1152    1
1153    1       CALL = MFSUM_INIT();                                    /* Perform init for this m.f. summary request */
1154    1       IF STATUS = NOT_SUCCESSFUL THEN CALL SIGNAL_MON_ERR();  /* Signal error if failure */
1155    1
1156  : 1       /*
1157  : 1       /*      Establish CTRL-C and CTRL-Z handlers for terminating the MONITOR request.
1158  : 1       /*      CTRL-C causes a MONITOR> prompt. CTRL-Z returns to DCL.
1159  : 1       /*/
1160    1
1161    1       CALL = ESTAB_CTRLCZ();                                  /* Establish CTRL-C and CTRL-Z handlers */
1162  : 1                                                               /* If error, do not terminate; simply ignore CTRL-C'
1163    1
1164  : 1       /*
1165  : 1       /*      For each input file, open it, do some checking of structure level, and classes,
1166  : 1       /*      and build the Column Summary Block (CSB) (if necessary) to represent the column
1167  : 1       /*      required in the summary report. There will be one column per file, or, if a
1168  : 1       /*      by-node report was requested, one column per node.
1169  : 1       /*/
1170    1
1171    1       NON_EMPTY_FILES = 0;                                    /* Init count of non-empty input files */
1172    1       DO I = 1 TO M->MRB$B_INP_FILES  WHILE(CTRLCZ_HIT = NO); /* Loop once per input file */
1173  : 2                                                               /* ... as long as user has not terminated */
1174    2       IFBPTR = ADDR(AN_IFB(I));                               /* Establish current Input File Block */
1175    2       M->MRB$A_INPUT = IFB$A_INPUT;                           /* Establish current input file */
1176    2       MC->MCA$V_EOF = NO;                                     /* End-of-file not hit yet */
1177    2
1178    2       CALL = INPUT_INIT();                                    /* Open input file and establish H as header record
1179    2       IF STATUS = NOT_SUCCESSFUL THEN                         /* If failed, */
1180    2         IF CALL ^= MNR$_PREMEOF THEN CALL SIGNAL_MON_ERR();   /*   then signal error for all but PREMEOF (empty fi
1181    2         ELSE DO;                                              /*   but for PREMEOF, */
1182    3              IFB$B_COL_NO = 0;                                /*   zero out column number, */
1183    3              CALL LINK_MON_ERR(MNR$_IGNFIL,IFB$A_INPUT);      /*   ... and issue info msg */
1184    3              END;
1185    2
1186    2       ELSE                                                    /* If successful, */
1187    2         DO;                                                   /* Beginning of successful do-group */
1188    3         NON_EMPTY_FILES = NON_EMPTY_FILES + 1;                /* Count this good file */
1189    3         IF NON_EMPTY_FILES = 1                                /* If first file, */
1190    3             THEN ST_LEV_COMM = H->MNR_HDR$T_LEVEL;            /* ... then establish common structure level */
1191    3             ELSE IF ST_LEV_COMM ^= H->MNR_HDR$T_LEVEL         /* ... otherwise, file level must match common */
1192    3                     THEN DO;
1193    4                          CALL MON_ERR(MNR$_NOCOMMSTLEV);      /* ... if not, it's an error */
1194    4                          CALL SIGNAL_MON_ERR();               /* ... signal it */
1195    4                          END;
1196    3
1197    3         IF MNR_HDR$K_CLASSBITS < MC->MCA$L_INPUT_LEN          /* If CLASSBITS field is defined for input file, */
1198    3             THEN FILE_CLASSES = H->MNR_HDR$O_CLASSBITS;       /*   then get file classes from usual place */
1199    3             ELSE FILE_CLASSES = H->MNR_HDR$O_REVOCLSBITS;     /*   else get them from another place */
1200  : 3                                                               /* NOTE -- MNR_HDR$O_REVOCLSBITS is used for compati
1201  : 3                                                               /*         with MONSC001 and MONBA001 file struct le
1202    3
1203    3         ALL_FILE_CLASSES = BOOL(ALL_FILE_CLASSES,FILE_CLASSES,OR_OP); /* Augment class string with class bits from this fi
1204    3
1205    3         CALL = GET_NODENAME();                                /* Get node name from current file */
```

```
1206    3       IF STATUS = NOT_SUCCESSFUL THEN CALL SIGNAL_MON_ERR();          /* Signal error if failure */
1207    3
```

```
1208 :  3       /*
1209 :  3       /*    Set up Column Summary Block (CSB)
1210 :  3       /*/
1211    3
1212    3       IF ^ M->MRB$V_BY_NODE ! NON_EMPTY_FILES = 1                     /* If not by-node request, or first input file */
1213    3         THEN CALL CREATE_NEW_CSB();                                   /*   then create a new CSB for a new column */
1214    3         ELSE DO;                                                      /*   else go look for a column with same node */
1215 :  4                                                                       /*        name as input file */
1216    4              FOUND_NODE = NO;                                         /* Indicate haven't found a matching node yet */
1217    4              DO J = 1 TO MFS$B_COLUMNS WHILE (FOUND_NODE = NO);       /* Loop once per CSB until found one */
1218    5                CSBPTR = CSB_POINTER(J);                               /* Get next CSB pointer */
1219    5                IF CURR_NODENAME = CSB$T_NODENAME                      /* If CSB node same as input file node, */
1220    5                    THEN DO;
1221    6                         IFB$B_COL_NO = J;                           /* Stash column no. in IFB for later use */
1222    6                         FOUND_NODE = YES;                           /* Indicate found a matching node */
1223    6                         END;
1224    5              END;
1225    4              IF FOUND_NODE = NO                                       /* If no matching node found in all CSBs, */
1226    4                  THEN CALL CREATE_NEW_CSB();                          /*   then create a new one */
1227    4              END;
1228    3         END;                                                          /* End of successful do-group */
1229    2
1230 :  2     /*
1231 :  2     /*      Perform cleanup operations on this input file (close it and free buffer).
1232 :  2     /*/
1233    2
1234    2     IF M->MRB$V_INP_CL_REQ                                          /* If cleanup required for this file, */
1235    2         THEN CALL INPUT_CLEANUP();                                  /* ... then do cleanup for it */
1236    2
1237    2     END;
1238    1
1239    1     IF NON_EMPTY_FILES = 0                                          /* If all files are empty, */
1240    1         THEN DO;
1241    2              CALL MON_ERR(MNR$_NOINPFILES);                         /*   then log the error */
1242    2              CALL SIGNAL_MON_ERR();                                 /*   ... and signal it */
1243    2              END;
1244    1
1245 :  1     /*
1246 :  1     /*      Eliminate unnecessary classes and set up Current Class Descriptor Array
1247 :  1     /*/
1248    1
1249    1     IF CTRLCZ_HIT = NO                                              /* If request is still running, */
1250    1         THEN DO;
1251    2              MFS$O_CLASSBITS = BOOL(ALL_FILE_CLASSES,MFS$O_CLASSBITS,AND_OP); /* Eliminate requested classes not in inpu
1252    2              IF MFS$O_CLASSBITS = '0'B                              /* If no classes left to do, */
1253    2                  THEN DO;
1254    3                       CALL MON_ERR(MNR$_NOCLASS);                   /*   then log error */
1255    3                       CALL SIGNAL_MON_ERR();                        /* ... and signal it */
1256    3                       END;
1257    2
1258    2
1259 :  2              /* NOTE ***** steal code to print warning message if one or more classes missing from the host of input fil
1260    2
1261 :  2              /*
1262 :  2              /*     Establish Current Class Descriptor Array
1263 :  2              /*/
```

```
1264    2
1265    2           CALL = ESTAB_CCD();                                 /* Set up CCD */
1266    2           IF STATUS = NOT_SUCCESSFUL THEN CALL SIGNAL_MON_ERR();  /* Signal error if failure */
1267    2
1268    2           END;
1269    1
```

```
1270    1
1271    1       /*
1272    1       /*      Main processing loop. For each input file, call EXECUTE_REQUEST
1273    1       /*      to perform the MONITOR request for that file. After each request
1274    1       /*      has terminated, the SUM buffer for each class will have been stored
1275    1       /*      in the multi-file summary buffer.
1276    1       /*      Then, when all input files have been processed, execute MF_SUMMARY_EVENT
1277    1       /*      to transform the multi-file summary buffer into the summary report.
1278    1       /*/
1279    1
1280    1       /*
1281    1       /*      Temporarily de-establish CTRL/C and CTRL/Z handlers, since they will be
1282    1       /*      established during each call to EXECUTE_REQUEST.
1283    1       /*/
1284    1
1285    1       IF CTRLCZ_HIT = NO & CTRLCZ_CHAN ^= 0                   /* If request is still running, */
1286    1           THEN DO;
1287    2                CALL = SYS$CANCEL(CTRLCZ_CHAN);                /* Cancel CTRL-C and CTRL-Z handlers */
1288    2                CTRLCZ_CHAN = 0;                               /* ... and indicate their disestablishment */
1289    2                END;
1290    1
1291    1       DO I = 1 TO M->MRB$B_INP_FILES  WHILE(CTRLCZ_HIT = NO); /* Loop once for each input file */
1292    2
1293    2       IFBPTR = ADDR(AN_IFB(I));                               /* Get next IFB pointer */
1294    2       IF IFB$B_COL_NO ^= 0                                    /* If this is a non-empty file, */
1295    2           THEN DO;
1296    3                MFS$B_CUR_COL = IFB$B_COL_NO;                          /* Set up current column no. for use by EXEC
1297    3                M->MRB$A_INPUT = IFB$A_INPUT;                          /* Set up MRB to refer to current input file
1298    3                M->MRB$Q_BEGINNING = MFS$Q_BEGINNING;                  /* Set up requested beginning time */
1299    3                M->MRB$Q_ENDING = MFS$Q_ENDING;                        /* Set up requested ending time */
1300    3                M->MRB$O_CLASSBITS = MFS$O_CLASSBITS;                  /* Start off each file with requested classe
1301    3
1302    3                CALL = EXECUTE_REQUEST();                              /* Execute MONITOR request for this input fi
1303    3                IF STATUS = NOT_SUCCESSFUL                             /* If request failed, */
1304    3                THEN CALL LINK_MON_ERR(MNR$_IGNFIL,IFB$A_INPUT);       /* Let user know we ignored a file */
1305    3
1306    3                ELSE DO;                                               /* Request successful ... check begin and en
1307    4                     CSBPTR = CSB_POINTER(IFB$B_COL_NO);               /* Get CSB pointer for current column */
1308    4                     CSB$B_FILES = CSB$B_FILES + 1;                    /* Count this file in this column */
1309    4                     IF ^ M->MRB$V_BY_NODE | QUAD_LT_QUAD(M->MRB$Q_BEGINNING,CSB$Q_BEGINNING) /* If new beginning establish
1310    4                       THEN CSB$Q_BEGINNING = M->MRB$Q_BEGINNING ;                    /* ... store it into CSB */
1311    4                     IF ^ M->MRB$V_BY_NODE | QUAD_LT_QUAD(CSB$Q_ENDING,M->MRB$Q_ENDING) /* If new ending established, */
1312    4                       THEN CSB$Q_ENDING = M->MRB$Q_ENDING ;                        /* ... store it into CSB */
1313    4                     END;
1314    3                END;
1315    2       END;
1316    1
```

```
1317    1       IF CTRLCZ_HIT = NO                                      /* If user hasn't hit CTRL/C or /Z, */
1318    1           THEN CALL = ESTAB_CTRLCZ();                         /*   then re-establish CTRL-C and CTRL-Z handlers */
1319    1                                                               /* If error, do not terminate; simply ignore CTRL-C'
1320    1       /*
1321    1       /*      Summarize all accumulated data into a single report.
1322    1       /*/
1323    1
1324    1       IF CTRLCZ_HIT = NO                                      /* If user is still patient, */
1325    1           THEN DO;
1326    2                CALL = MF_SUMMARY_EVENT();                     /* Output entire summary buffer for all classes */
1327    2                IF STATUS = NOT_SUCCESSFUL THEN CALL SIGNAL_MON_ERR(); /* Signal error if failure */
1328    2                END;
1329    1
1330    1       /*
1331    1       /*      Perform cleanup operations for this M.F. Summary request
1332    1       /*/
1333    1
1334    1       CALL MF_REQ_CLEANUP();                                  /* Execute various cleanup routines */
1335    1
1336    1       /*
1337    1       /*      Exit from MFSUM_REQUEST routine.
1338    1       /*      Note -- we get to this point either by falling through
1339    1       /*      the above code (normal path), or by direct branch from
1340    1       /*      the condition-handling routine (error path).
1341    1       /*/
1342    1
1343    1       MF_REQ_STATUS = NORMAL;                                 /* Normal status if we get to this point */
1344    1
1345    1       MF_REQ_EXIT:
1346    1
1347    1       RETURN(MF_REQ_STATUS);                                  /* Return to MONMAIN with completion status */
1348    1
1349    1
```

```
1350    1       MFSUM_INIT: Procedure Returns(Fixed Binary(31));                /* Initialization for M.F. Summary Request */
1351    2
1352    2       Declare
1353    2       CLASSES BIT(128),                                               /* Temporary class bit string */
1354    2       CLASSES_VEC (0:127) BIT(1) DEFINED(CLASSES);                    /* Bit-addressable alias */
1355    2
1356    2       CTRLCZ_HIT = NO;                                                /* Indicate CTRL-C and CTRL-Z not hit */
1357    2       CTRLCZ_CHAN = 0;                                                /* ... and no channel assigned for them */
1358    2
1359    2       CSBVEC_PTR = ADDR(CSBVEC);                                      /* Stash address of CSB Vector for later use */
1360    2
1361    2       ALLOCATE MFS;                                                   /* Allocate space for Multi-File Summary block */
1362    2
1363    2       MFS$B_COLUMNS = 0;                                              /* Start out request with no columns */
1364    2
1365    2       ALL_FILE_CLASSES = '0'B;                                        /* Start out request with no input file classes */
1366    2
1367    2       /*
1368    2       /*      Save requested BEGINNING and ENDING times in the MFS so they
1369    2       /*      can later be re-loaded to the MRB as EXECUTE_REQUEST is called
1370    2       /*      for each input file.
1371    2       /*/
1372    2
1373    2       MFS$Q_BEGINNING = M->MRB$Q_BEGINNING;                           /* Save beginning time */
1374    2       MFS$Q_ENDING = M->MRB$Q_ENDING;                                 /* Save ending time */
1375    2
1376    2       MFS$A_IFB_TAB = M->MRB$A_INPUT;                                 /* Save pointer to IFB Table for later use */
1377    2       IFB_TAB_PTR = MFS$A_IFB_TAB;                                    /* Set up IFB Table */
1378    2
1379    2       /*
1380    2       /*      Obtain space for statistics buffers
1381    2       /*/
1382    2
1383    2       ALLOCATE BUFFERS;                                               /* Get the space and set up MFS$A_STATSBUF */
1384    2       MFS$L_STATSBUF = 4 * 3 * MAXELTS_MFS;                           /* Compute and save its length (in bytes) */
1385    2
1386    2       /*
1387    2       /*      Turn off class bits for classes not eligible for
1388    2       /*      multi-file summary.
1389    2       /*/
1390    2
1391    2       CLASSES = M->MRB$O_CLASSBITS;                                   /* Get list of requested classes */
1392    2       CLASSES_VEC(PROCS_CLSNO) = NO;                                  /* Turn off PROCESSES class bit */
1393    2
1394    2       MFS$O_CLASSBITS = CLASSES;                                      /* Move updated class bit string to the MFS */
1395    2
1396    2       IF MFS$O_CLASSBITS = '0'B                                       /* If no classes left to do, */
1397    2           THEN DO;
1398    3                CALL MON_ERR(MNR$_NOCLASSES);                          /*   then log error */
1399    3                RETURN(MNR$_NOCLASSES);                                /*   ... and return with status */
1400    3                END;
1401    2
```

```
1402  :  2        /*
1403  :  2        /*      Check for /SUMMARY qualifier specified, and return
1404  :  2        /*      error if not. Move summary file spec ptr from MRB to
1405  :  2        /*      MFS, and explicitly clear the /SUMMARY, /DISPLAY
1406  :  2        /*      and /RECORD file spec ptrs from the MRB.
1407  :  2        /*/
1408     2
1409     2        IF M->MRB$A_SUMMARY = NULL()                          /* If no summary qualifier, */
1410     2            THEN DO;
1411     3                 CALL MON_ERR(MNR$_NOSUMM);                   /*   then log the error */
1412     3                 RETURN(MNR$_NOSUMM);                         /*   ... and return with status */
1413     3                 END;
1414     2            ELSE DO;
1415     3                 MFS$A_SUMMARY = M->MRB$A_SUMMARY;            /* Move /SUMMARY file spec ptr to MFS */
1416     3                 M->MRB$A_SUMMARY = NULL();                   /* Indicate no regular /SUMMARY */
1417     3                 M->MRB$A_DISPLAY = NULL();                   /* Indicate no /DISPLAY */
1418     3                 M->MRB$A_RECORD = NULL();                    /* Indicate no /RECORD */
1419     3                 END;
1420     2
1421     2        RETURN(NORMAL);                                       /* Return */
1422     2        END MFSUM_INIT;
1423     1
```

```
1424     1        CREATE_NEW_CSB: Procedure;                          /* Create a new Column Summary Block */
1425     2
1426     2        MFS$B_COLUMNS = MFS$B_COLUMNS + 1;                  /* Count a new column */
1427     2        IFB$B_COL_NO = MFS$B_COLUMNS;                       /* ... and store its number in current IFB */
1428     2
1429     2        ALLOCATE CSB;                                       /* Get space for CSB */
1430     2        CSB_POINTER(MFS$B_COLUMNS) = CSBPTR;                /* Add its pointer to CSB vector */
1431     2        CSB$T_NODENAME = CURR_NODENAME;                     /* Get nodename from current file */
1432     2        CSB$Q_BEGINNING = 'FFFFFFFFFFFFFFFF'B4;             /* Init to latest beginning time */
1433     2        CSB$Q_ENDING = '0'B;                                /* ... and earliest ending time */
1434     2        CSB$B_FILES = 0;                                    /* Init no. of files in this column */
1435     2        CSB$V_IGNORE = NO;                                  /* Don't ignore this column */
1436     2
1437     2        RETURN;                                             /* Return */
1438     2        END CREATE_NEW_CSB;
1439     1
```

```
1440    1       GET_NODENAME: Procedure Returns(Fixed Binary(31));              /* Get nodename from current input file */
1441    2
1442    2       Declare
1443    2       READ_INPUT      ENTRY (FIXED BINARY(31)),                       /* MONITOR routine to read an input (playback) file
1444    2       NEXT_REC        FIXED BINARY(31) GLOBALREF VALUE;               /* Read next record indicator for READ_INPUT rtn */
1445    2
1446    2       Declare
1447    2       SYI_TYPE        FIXED BINARY(15) GLOBALREF,                     /* Type for MONITOR recording file sys info record *
1448    2       TEMP_TYPE       BIT(8) ALIGNED,                                 /* Temporary area for record type byte */
1449    2       SYIPTR          POINTER;                                        /* Pointer to sys info record */
1450    2
1451    2       CALL READ_INPUT(NEXT_REC);                                      /* Read system information record */
1452    2       IF MC->MCA$V_EOF                                                /* If end-of-file, */
1453    2           THEN DO;
1454    3                CALL MON_ERR(MNR$_PREMEOF);                            /* Can't find sys info record; log the error */
1455    3                RETURN (MNR$_PREMEOF);                                 /* ... and return to caller */
1456    3                END;
1457    2
1458    2       SYIPTR = MC->MCA$A_INPUT_PTR;                                   /* Establish ptr to sys info record */
1459    2       TEMP_TYPE = UNSPEC(SYI_TYPE);                                   /* Get sys info type into a byte for compare */
1460    2       IF SYIPTR->MNR_SYI$B_TYPE ^= TEMP_TYPE                          /* If this record is not the sys info rec, */
1461    2           THEN DO;
1462    3                CALL MON_ERR(MNR$_INVINPFIL);                          /* Log an error */
1463    3                RETURN(MNR$_INVINPFIL);                                /* ... and return to caller */
1464    3                END;
1465    2
1466    2       IF MNR_SYI$K_NODENAME < MC->MCA$L_INPUT_LEN                     /* If NODENAME field is defined for input file, */
1467    2           THEN CURR_NODENAME = SYIPTR->MNR_SYI$T_NODENAME;            /* ... then pick it up from there */
1468    2           ELSE UNSPEC(CURR_NODENAME) = '0'B;                          /* Otherwise, simply clear it */
1469    2
1470    2       RETURN(NORMAL);                                                 /* Return */
1471    2       END GET_NODENAME;
1472    1
```

```
1473    1       ESTAB_CCD: Procedure Returns(Fixed Binary(31));                 /* Establish Current Class Descriptor array */
1474    2
1475    2       /*
1476    2       /*      !-------------------------------------------------------------------------+
1477    2       /*      !                                                                         !
1478    2       /*      !                                LOCAL STORAGE                            !
1479    2       /*      !                                                                         !
1480    2       /*      !-------------------------------------------------------------------------+
1481    2       /*/
1482    2
1483    2       Declare
1484    2       DO_CLASSES      BIT(128),                                       /* Classes to do */
1485    2       DO_CLASSES_VEC (0:127) BIT(1) DEFINED(DO_CLASSES),              /* Bit-addressable alias */
1486    2       CLASS_NO        FIXED BINARY(7);                                /* Class number */
1487    2
1488    2       /*
1489    2       /*      Given MFS$O_CLASSBITS, execute do loop using INDEX builtin
1490    2       /*      to fill in the CCD (Current Class Descriptor) array.
1491    2       /*/
1492    2
1493    2       DO_CLASSES = MFS$O_CLASSBITS;                                   /* Get list of classes to do */
1494    2       CLASS_NO = 0;                                                   /* Initialize class number */
1495    2       DO I = 1 TO MAX_CLASS_NO + 1  WHILE(CLASS_NO >= 0);             /* Loop once for each possible class */
1496    3         CLASS_NO = INDEX(DO_CLASSES,YES) - 1;                         /* Find next requested class number */
1497    3         IF CLASS_NO >= 0                                              /* Only continue if a class was found */
1498    3           THEN DO;
1499    4                DO_CLASSES_VEC(CLASS_NO) = NO;                         /* Eliminate it from future consideration */
1500    4                CURR_CLASS_NO(I) = CLASS_NO;                           /* Store class_no in CCD table */
1501    4                MFS$Q_CLASSCT = I;                                     /* Keep track of class count */
1502    4                CURR_CDBPTR(I) = ADDR(CDBLOCK(CLASS_NO));              /* ... and store it in CCD table */
1503    4                DCDB = CURR_CDBPTR(I);                                 /* Get CDB address'ty */
1504    4                DCDB->CDB$A_SUMBUF = NULL();                           /* Make sure we get a new summary buffer */
1505    4                IF CURR_CDBPTR(I)->CDB$V_DISABLE = YES                 /* If this class is disabled, */
1506    4                    THEN DO;
1507    5                         CALL MON_ERR(MNR$_CLASDISAB);                 /* Log the error */
1508    5                         RETURN(MNR$_CLASDISAB);                       /* ... and return */
1509    5                         END;
1510    4                END;
1511    3       END;
1512    2
1513    2       RETURN(NORMAL);                                                 /* Normal return */
1514    2
1515    2       END ESTAB_CCD;
1516    1
```

```
1517    1       MF_SUMMARY_EVENT: Procedure Returns(Fixed Binary(31));          /* Multi-File Summary Event */
1518    2
1519    2       /*
1520    2       /*++
1521    2       /*
1522    2       /* FUNCTIONAL DESCRIPTION:
1523    2       /*
1524    2       /*      MF_SUMMARY_EVENT
1525    2       /*
1526    2       /*      Called by MFSUM_REQUEST once per request to create a
1527    2       /*      summary file containing one or more screen images for
1528    2       /*      each of the requested classes.
1529    2       /*
1530    2       /* INPUTS:
1531    2       /*
1532    2       /*      None
1533    2       /*
1534    2       /* OUTPUTS:
1535    2       /*
1536    2       /*      None
1537    2       /*
1538    2       /* ROUTINE VALUE:
1539    2       /*
1540    2       /*      SS$_NORMAL, or failing MONITOR status code.
1541    2       /*
1542    2       /*--
1543    2       /*/
1544    2
```

```
1545    2       /*
1546    2       /*      +-----------------------------------------------------------------------+
1547    2       /*      |                                                                       |
1548    2       /*      |                              LOCAL STORAGE                            |
1549    2       /*      |                                                                       |
1550    2       /*      +-----------------------------------------------------------------------+
1551    2       /*/
1552    2
1553    2       Declare
1554    2       CURR_CLASS              FIXED BINARY(15),               /* Consec no (not class no) of current class */
1555    2       FULL_PAGES              FIXED BINARY(7),                /* Number of full pages to display */
1556    2       PARTIAL_PAGE_COLS       FIXED BINARY(7),                /* Number of columns on the partial (last) page */
1557    2       COLS_PER_PAGE           FIXED BINARY(7) INIT(5);        /* Number of columns which fit on a single page */
1558    2
1559    2       Declare
1560    2       LARGE_NO        FIXED BINARY(31) GLOBALREF VALUE,       /* Very large number (integer or floating) */
1561    2       1 BUFFERS       BASED(MFS$A_STATSBUF),                  /* Statistics buffers */
1562    2         2 TOT (1:MAXELTS_MFS) FIXED BINARY(31),               /* TOTAL statistics buffer */
1563    2         2 MIN (1:MAXELTS_MFS) FIXED BINARY(31),               /* MINIMUM statistics buffer */
1564    2         2 MAX (1:MAXELTS_MFS) FIXED BINARY(31);               /* MAXIMUM statistics buffer */
1565    2
1566    2       Declare
1567    2       PUT_LEN         FIXED BINARY(31),                       /* Length of buffer for DISPLAY_PUT to put */
1568    2       1 DPUT_FLAGS,                                           /* DISPLAY_PUT routine flags */
1569    2         2 FAOL_REQUESTED BIT(8) ALIGNED,                      /* YES => Xlate buffer with FAOL first */
1570    2         2 OUTP_REQUESTED BIT(8) ALIGNED;                      /* YES => Really output buffer */
1571    2
1572    2       Declare
1573    2       DISPLAY_PUT     ENTRY(ANY, FIXED BINARY(31), ANY, ANY)  /* MACRO-32 rtn to put a display string */
1574    2                       OPTIONS(VARIABLE)
1575    2                       RETURNS(FIXED BINARY(31)),
1576    2       ADV_HOM_ITEM    ENTRY (POINTER);                        /* MACRO-32 rtn to advance homog class to next displ
1577    2
1578    2       Declare
1579    2       DCDB            POINTER STATIC;                         /* CDB for current class */
1580    2
1581    2       Declare
1582    2       SPTR            POINTER GLOBALREF;                      /* Pointer to SYI (System Information Area) */
1583    2
```

```
1584    2
1585    2       UNSPEC(SPTR->MNR_SYI$T_NODENAME) = '0'B;                        /* Clear nodename to avoid misleading node in headin
1586    2
1587    2       /*
1588    2       /*      Loop through CSBs (Column Summary Blocks), turning on CSB$V_IGNORE
1589    2       /*      for each one with CSB$Q_ENDING = 0. These are columns which never
1590    2       /*      accumulated any data. If all columns are ignored, terminate the
1591    2       /*      MONITOR request. Establish new value for MFS$B_COLUMNS which is
1592    2       /*      the new column count, excluding ignored columns.
1593    2       /*/
1594    2
1595    2       J = 0;                                                          /* Init count of ignored columns */
1596    2       DO I = 1 TO MFS$B_COLUMNS;                                      /* Loop once for each column */
1597    3       CSBPTR = CSB_POINTER(I);                                        /* Get CSB pointer for this column */
1598    3       IF CSB$Q_ENDING = '0'B                                          /* If no data for this column, */
1599    3           THEN DO;
1600    4                CSB$V_IGNORE = YES;                                    /*   ignore it, */
1601    4                J = J + 1;                                             /*   ... and count it */
1602    4                END;
1603    3       END;
1604    2
1605    2       MFS$B_COLUMNS = MFS$B_COLUMNS - J;                              /* Compute new column count */
1606    2       IF MFS$B_COLUMNS = 0                                            /* If no columns, */
1607    2           THEN DO;
1608    3                CALL MON_ERR(MNR$_NOINPFILES);                         /*   then log the error */
1609    3                RETURN(MNR$_NOINPFILES);                               /*   ... and return with status */
1610    3                END;
1611    2
1612    2       M->MRB$A_SUMMARY = MFS$A_SUMMARY;                               /* Get /SUMMARY file spec ptr to MRB for call */
1613    2       CALL = SUMMARY_INIT();                                          /* Do summary file init */
1614    2       IF STATUS = NOT_SUCCESSFUL                                      /* Failed? */
1615    2           THEN DO;
1616    3                CALL MON_ERR(MNR$_DISPERR,CALL);                       /* Yes -- log the error */
1617    3                RETURN(MNR$_DISPERR);                                  /* ... and return with status */
1618    3                END;
1619    2
1620    2       FULL_PAGES = DIVIDE(MFS$B_COLUMNS,COLS_PER_PAGE,7);             /* Compute no. of full report pages of data */
1621    2       PARTIAL_PAGE_COLS = MFS$B_COLUMNS - (FULL_PAGES * COLS_PER_PAGE); /* ... and number of partially filled pages */
1622    2
1623    2
```

```
1624  1  2      /*
1625  1  2      /*      Summarize each class individually.
1626  1  2      /*/
1627     2
1628     2      DO CURR_CLASS = 1 TO MFS$W_CLASSCT;                          /* Loop once for each requested class */
1629     3
1630     3      DCDB = CURR_CDBPTR(CURR_CLASS);                              /* Get CDB ptr for current class */
1631     3
1632     3      IF DCDB->CDB$V_HOMOG                                         /* If homogeneous class, */
1633     3          THEN MFS$L_ELEMS = DCDB->CDB$A_CDX->CDX$L_DCOUNT;        /*  then set this many elements */
1634     3          ELSE IF DCDB->CDB$V_SYSCLS                               /*  else if SYSTEM class, */
1635     3                  THEN DO;
1636     4                       MFS$L_ELEMS = ECOUNT_SYS_ALL;               /*       then use SYS/ALL element count */
1637     4                       DCDB->CDB$B_ST = ALL_STAT;                  /*            and force a tabular display */
1638     4                       END;
1639     3                  ELSE MFS$L_ELEMS = DCDB->CDB$L_ECOUNT;           /*  this many elements for other heteros */
1640     3
1641     3      IF DCDB->CDB$V_HOMOG                                         /* If homogeneous class, */
1642     3          THEN DO;
1643     4               DCDB->CDB$A_CDX->CDX$B_IDISCONSEC = 0;              /* Init consec display item number */
1644     4                 DO WHILE(DCDB->CDB$A_CDX->CDX$B_IDISCONSEC < DCDB->CDB$A_CDX->CDX$B_IDISCT);
1645     5                 CALL ADV_HOM_ITEM(DCDB);                          /* Advance to next display item */
1646     5                 CALL = SUMM_ONE_CLASS();                          /* Summarize once for each item */
1647     5                 IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL);    /* Check call */
1648     5                 END;
1649     4               END;
1650     3
1651     3          ELSE DO;                                                 /* Heterogeneous class or PROCESSES */
1652     4               CALL = SUMM_ONE_CLASS();                            /* Only need to call once */
1653     4               IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL);      /* Check call */
1654     4               END;
1655     3
1656     3      END;
1657     2
1658     2      RETURN(NORMAL);                                              /* Return */
1659     2
```

```
1660    2       SUMM_ONE_CLASS: Procedure Returns(Fixed Binary(31));            /* Perform m.f. summary output for a single class */
1661    3
1662    3       /*
1663    3       /*++
1664    3       /*
1665    3       /* FUNCTIONAL DESCRIPTION:
1666    3       /*
1667    3       /*      SUMM_ONE_CLASS
1668    3       /*
1669    3       /*      Called by MF_SUMMARY_EVENT to put screen images to the
1670    3       /*      summary file for a single class.
1671    3       /*
1672    3       /* INPUTS:
1673    3       /*
1674    3       /*      None
1675    3       /*
1676    3       /* OUTPUTS:
1677    3       /*
1678    3       /*      None
1679    3       /*
1680    3       /* ROUTINE VALUE:
1681    3       /*
1682    3       /*      SS$_NORMAL, or failing MONITOR status code.
1683    3       /*
1684    3       /*--
1685    3
1686    3       /*/
1687    3       /*
1688    3       /*      +-----------------------------------------------------------------------+
1689    3       /*      |                                                                       |
1690    3       /*      |                          LOCAL STORAGE                                |
1691    3       /*      |                                                                       |
1692    3       /*      +-----------------------------------------------------------------------+
1693    3       /*/
1694    3
1695    3       Declare
1696    3       PAGES           FIXED BINARY(15);                               /* No. of full pages to display (loop control) */
1697    3
1698    3       Declare
1699    3       START_COL       FIXED BINARY(7),                                /* Starting column number */
1700    3       COLS_TO_DO      FIXED BINARY(7),                                /* Number of columns to display */
1701    3       END_COL         FIXED BINARY(7),                                /* Ending column number */
1702    3       STAT_IND        BIT(1) ALIGNED;                                 /* Statistics Indicator -- YES => display statistics
1703    3
```

```
1704    3
1705    3       /*
1706    3       /*      Initialize statistics (TOT, MIN, and MAX) buffers.
1707    3       /*/
1708    3
1709    3       DO I = 1 TO MFS$L_ELEMS;                                        /* Loop once per element */
1710    4       TOT(I) = 0;                                                     /* Clear total to 0 */
1711    4       MIN(I) = LARGE_NO;                                              /* Set min to largest number */
1712    4       MAX(I) = 0;                                                     /* Set max to smallest number */
1713    4       END;
1714    3
1715    3       MFS$B_DATA_COLS = 0;                                            /* Init no. of columns of data (used by FILL_MFSUM_F
1716    3
1717    3       /*
1718    3       /*      Put as many full pages of variable information to summary file as
1719    3       /*      required by this class.
1720    3       /*/
1721    3
1722    3       START_COL = 1;                                                  /* Start with column 1 */
1723    3
1724    3       DO PAGES = 1 TO FULL_PAGES;                                     /* Loop once for each full page */
1725    4       CALL = PUT_SUMM_PAGE(START_COL,COLS_PER_PAGE,END_COL,NO);       /* Put a single page for this class */
1726    4       IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL);                   /* Check status */
1727    4       START_COL = END_COL + 1;                                        /* Establish new starting column for next page */
1728    4       END;
1729    3
1730    3       /*
1731    3       /*      Now put one or, possibly, two partial pages of variable information
1732    3       /*      to the summary file for this class.
1733    3       /*/
1734    3
1735    3       IF PARTIAL_PAGE_COLS <= 4                                       /* If no more than 4 columns on partial page, */
1736    3           THEN DO;                                                    /* ... then can fit the statistics on same page */
1737    4                CALL = PUT_SUMM_PAGE(START_COL,PARTIAL_PAGE_COLS,END_COL,YES); /* Put a single page for this class */
1738    4                IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL);          /* Check status */
1739    4                END;
1740    3
1741    3           ELSE DO;                                                    /* ... otherwise, one page for data and one for stat
1742    4                CALL = PUT_SUMM_PAGE(START_COL,PARTIAL_PAGE_COLS,END_COL,NO); /* Put a single page for this class */
1743    4                IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL);          /* Check status */
1744    4
1745    4                CALL = PUT_SUMM_PAGE(START_COL,0,END_COL,YES);         /* Put a single page for this class */
1746    4                IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL);          /* Check status */
1747    4                END;
1748    3
1749    3
1750    3       RETURN(NORMAL);
1751    3
1752    3       END SUMM_ONE_CLASS;
1753    2
```

```
1754    2       PUT_SUMM_PAGE: Procedure (START_COL,COLS_TO_DO,END_COL,STAT_IND)        /* Output a single summary page for curr cla
1755    2                      Returns(Fixed Binary(31));
1756    2
1757    3       /*
1758    3       /*++
1759    3       /*
1760    3       /* FUNCTIONAL DESCRIPTION:
1761    3       /*
1762    3       /*      PUT_SUMM_PAGE
1763    3       /*
1764    3       /*      TBS
1765    3       /*
1766    3       /* INPUTS:
1767    3       /*
1768    3       /*      START_COL  -- Column number of 1st column to display.
1769    3       /*
1770    3       /*      COLS_TO_DO -- Number of columns to display.
1771    3       /*
1772    3       /*      STAT_IND   -- Flag indicating whether or not to display statistics.
1773    3       /*                    ON means "display statistics".
1774    3       /*                    OFF means "don't display statistics".
1775    3       /*
1776    3       /* OUTPUTS:
1777    3       /*
1778    3       /*      END_COL    -- Column number of last column displayed.
1779    3       /*
1780    3       /* ROUTINE VALUE:
1781    3       /*
1782    3       /*      SS$_NORMAL, or failing MONITOR status code.
1783    3       /*
1784    3       /*--
1785    3       /*/
1786    3
```

```
1787  3       /*
1788  3       /*      +-----------------------------------------------------------------------+
1789  3       /*      |                                                                       |
1790  3       /*      |                          LOCAL STORAGE                                |
1791  3       /*      |                                                                       |
1792  3       /*      +-----------------------------------------------------------------------+
1793  3       /*/
1794  3
1795  3       Declare
1796  3       START_COL       FIXED BINARY(7),                        /* Starting column number */
1797  3       COLS_TO_DO      FIXED BINARY(7),                        /* Number of columns to display */
1798  3       END_COL         FIXED BINARY(7),                        /* Ending column number */
1799  3       STAT_IND        BIT(1) ALIGNED;                         /* Statistics Indicator -- YES => display statistics
1800  3
1801  3       Declare
1802  3       DISP_TEMPLATE   ENTRY (POINTER, BIT(1) ALIGNED)         /* Rtn to display the summary template */
1803  3                       RETURNS (FIXED BINARY(31)),
1804  3       FILL_MFSUM_FAOSTK ENTRY(POINTER, FIXED BINARY(7), FIXED BINARY(7), BIT(1) ALIGNED)
1805  3                       RETURNS (FIXED BINARY(31)),             /* MACRO-32 rtn to fill the FAOSTK with data */
1806  3       DISPLAY_HOMOG   ENTRY (POINTER)                         /* MACRO-32 rtn to display homog class data */
1807  3                       RETURNS (FIXED BINARY(31));
1808  3
1809  3       Declare
1810  3       1 MF_SUMM1_STR  GLOBALREF,                              /* M.F. Summary fixed FAO control string */
1811  3         2 L           FIXED BINARY(7),                        /* Length */
1812  3         2 S           CHAR(1);                                /* First character of string */
1813  3
1814  3       Declare
1815  3       DATA_STR        CHAR(1) BASED(DCDB->CDB$A_FAOCTR),      /* First char of FAO ctr str for summary data */
1816  3       FAOSTK          FIXED BINARY(31) GLOBALREF;             /* First longword of FAOL parm list */
1817  3
1818  3
```

```
1819    ;
1820    ;       /*
1821    ;       /*      Fill FAOSTK with data for this page for this class
1822    ;       /*/
1823    ;
1824            CALL = FILL_MFSUM_FAOSTK(DCDB,START_COL,COLS_TO_DO,STAT_IND);   /* Fill FAOSTK with data */
1825            IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL);                   /* Check status */
1826    ;
1827    ;       /*
1828    ;       /*      Put out template, including utility identification, class name,
1829    ;       /*      element labels for hetero classes, etc.
1830    ;       /*/
1831    ;
1832            CALL = DISP_TEMPLATE(DCDB,NO);                                  /* Send template to SCRPKG, but don't output yet */
1833            IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL);                  /* Check call */
1834    ;
1835    ;       /*
1836    ;       /*      Put out column headings, including node name, beginning and
1837    ;       /*      ending times, and statistics headings (TOT, AVE, MIN, MAX) if requested.
1838    ;       /*/
1839    ;
1840    ;       /*
1841    ;       /*      Start with fixed portions of "screen."
1842    ;       /*
1843    ;       /*/
1844    ;
1845            PUT_LEN = MF_SUMM1_STR.L;                                       /* Length of fixed summary control string */
1846            FAOL_REQUESTED = YES;                                           /* Run it through FAOL */
1847            OUTP_REQUESTED = NO;                                            /* ... but don't output it yet */
1848            CALL = DISPLAY_PUT(DPUT_FLAGS,PUT_LEN,MF_SUMM1_STR.S,);         /* Send summary line to SCRPKG */
1849            IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL);                   /* Check status */
1850    ;
1851    ;       /*
1852    ;       /*      Next display node name column headings
1853    ;       /*/
1854    ;
1855            CALL = MFSUM_HEADINGS(START_COL,COLS_TO_DO,END_COL,STAT_IND);   /* Display column headings */
1856            IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL);                   /* Check status */
1857    ;
1858    ;       /*
1859    ;       /*      Put a screenful of data to summary file
1860    ;       /*/
1861    ;
1862            IF DCDB->CDB$V_HOMOG                                            /* Check type of standard class */
1863                THEN DO;                                                    /* Homogeneous Standard Class */
1864                     CALL = DISPLAY_HOMOG(DCDB);                            /* Send homog data display lines to SCRPKG */
1865                     IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL);          /* Check status */
1866                     END;
1867                ELSE DO;                                                    /* Heterogeneous Standard Class */
1868                     FAOL_REQUESTED = YES;                                  /* Run it through FAOL */
1869                     OUTP_REQUESTED = YES;                                  /* Output it now */
1870                     CALL = DISPLAY_PUT(DPUT_FLAGS,DCDB->CDB$L_FAOCTR,DATA_STR,FAOSTK);
1871    ;                                                                       /* Send display data to SCRPKG */
1872                     IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL);          /* Check status */
1873                     END;
1874
```

```
 1875    3       RETURN(NORMAL);                 /* Return */
 1876    3
 1877    3       END PUT_SUMM_PAGE;
 1878    2
```

```
1879    2       MFSUM_HEADINGS: Procedure (START_COL,COLS_TO_DO,END_COL,STAT_IND) /* Output column headings for curr class and page
1880                    Returns(Fixed Binary(31));
1881
1882    3       /*
1883    3       /*++
1884    3       /*
1885    3       /* FUNCTIONAL DESCRIPTION:
1886    3       /*
1887    3       /*      MFSUM_HEADINGS
1888    3       /*
1889    3       /*      Send to SCRPKG a column heading for each of the requested columns
1890    3       /*      for the current output page and class. The heading consists of a
1891    3       /*      nodename and beginning and ending times.
1892    3       /*
1893    3       /* INPUTS:
1894    3       /*
1895    3       /*      START_COL  -- Column number of 1st column to display.
1896    3       /*
1897    3       /*      COLS_TO_DO -- Number of columns to display.
1898    3       /*
1899    3       /*      STAT_IND   -- Flag indicating whether or not to display statistics.
1900    3       /*                    ON means "display statistics headings".
1901    3       /*                    OFF means "don't display statistics headings".
1902    3       /*
1903    3       /* OUTPUTS:
1904    3       /*
1905    3       /*      END_COL    -- Column number of last column displayed.
1906    3       /*
1907    3       /*
1908    3       /* ROUTINE VALUE:
1909    3       /*
1910    3       /*      SS$_NORMAL, or failing MONITOR status code.
1911    3       /*
1912    3       /*--
1913    3       /*/
1914    3
```

```
: 3      /*
: 3      /*      +--------------------------------------------------------------+
: 3      /*      |                                                              |
: 3      /*      |                       LOCAL STORAGE                          |
: 3      /*      |                                                              |
: 3      /*      +--------------------------------------------------------------+
: 3      /*/
  3
  3      Declare
  3      MFS_NOD_SEGLEN  FIXED BINARY(31) GLOBALREF VALUE,           /* Length of a segment of node control string */
  3      MFS_TIM_SEGLEN  FIXED BINARY(31) GLOBALREF VALUE;           /* Length of a segment of time control string */
  3
  3      Declare
  3      START_COL       FIXED BINARY(7),                            /* Starting column number */
  3      COLS_TO_DO      FIXED BINARY(7),                            /* Number of columns to display */
  3      END_COL         FIXED BINARY(7),                            /* Ending column number */
  3      STAT_IND        BIT(1) ALIGNED;                             /* Statistics Indicator -- YES => display statistics
  3
  3      Declare
  3      STHEAD_COL      FIXED BINARY(7);                            /* Starting column no. for stats heading */
  3
  3      Declare
  3      SCR$SET_CURSOR  ENTRY (ANY VALUE, ANY VALUE);               /* SCRPKG rtn to set the cursor position */
  3
  3      Declare
  3      1 MFS_NODE_PARMS,                                           /* FAOL parms for nodenames */
  3       2  MFS_NODE_P (1:COLS_PER_PAGE),
  3        3 MFS_NODE_PARM POINTER,                                  /* Node name parameter */
  3        3 MFS_FIELD_PARM FIXED BINARY(31),                        /* "Number of files" field width */
  3        3 MFS_FILES_PARM FIXED BINARY(31);                        /* "Number of files" parameter */
  3
  3
  3      Declare
  3      1 MFS_BEG_PARMS,                                            /* FAOL parms for beginning times */
  3        2 MFS_BEG_PARM (1:COLS_PER_PAGE) POINTER;                 /* One parm per column */
  3
  3      Declare
  3      1 MFS_END_PARMS,                                            /* FAOL parms for ending times */
  3        2 MFS_END_PARM (1:COLS_PER_PAGE) POINTER;                 /* One parm per column */
  3
  3      Declare
  3      1 MFS_NODE_STR  GLOBALREF,                                  /* M.F. Summary nodename FAO control string */
  3        2 L           FIXED BINARY(7),                            /* Length */
  3        2 S           CHAR(1);                                    /* First character of string */
  3
  3      Declare
  3      1 MFS_TIME_STR  GLOBALREF,                                  /* M.F. Summary beg/end time FAO control string */
  3        2 L           FIXED BINARY(7),                            /* Length */
  3        2 S           CHAR(1);                                    /* First character of string */
  3
  3      Declare
  3      1 MFS_STHEAD1_STR GLOBALREF,                                /* M.F. Summary stats head FAO control string (1st)
  3        2 L           FIXED BINARY(7),                            /* Length */
  3        2 S           CHAR(1);                                    /* First character of string */
  3
  3      Declare
```

```
1971    3       1 MFS_STHEAD2_STR GLOBALREF,                    /* M.F. Summary stats head FAO control string (2nd)
1972    3         2 L             FIXED BINARY(7),              /* Length */
1973    3         2 S             CHAR(1);                      /* First character of string */
1974    3
1975    3
```

```
1976    3       IF COLS_TO_DO ^= 0                                      /* If there is some work to do, */
1977    3           THEN DO;
1978    4
1979    4                   /*
1980    4                   /*      Fill parameter lists for nodename and beginning and ending times.
1981    4                   /*/
1982    4
1983    4                   J = 0;                                      /* Clear count of columns actually processed */
1984    4
1985    4                   DO I = START_COL TO MAX_INP_FILES  WHILE(J ^= COLS_TO_DO); /* Keep looking at columns until all processed *
1986    5                   END_COL = I;                                /* Remember ending column number for caller */
1987    5                   CSBPTR = CSB_POINTER(I);                    /* Get next CSB (Column Summary Block) ptr */
1988    5                   IF CSB$V_IGNORE = NO                        /* If it is actually to be processed, */
1989    5                       THEN DO;
1990    6                            J = J + 1;                         /* Count this column */
1991    6                            MFS_NODE_PARM(J) = ADDR(CSB$T_NODENAME); /* Move nodename parm to list */
1992    6                            MFS_FILES_PARM(J) = CSB$B_FILES;   /* Move "number of files" parm to list */
1993    6                            IF CSB$B_FILES <= 1                /* If 1 or no files, */
1994    6                                THEN MFS_FIELD_PARM(J) = 0;    /*   then field width = 0 (no display) */
1995    6                                ELSE MFS_FIELD_PARM(J) = 5;    /*   otherwise, field width = 5 */
1996    6                            MFS_BEG_PARM(J) = ADDR(CSB$Q_BEGINNING); /* Move beginning time parm to list */
1997    6                            MFS_END_PARM(J) = ADDR(CSB$Q_ENDING); /* Move ending time parm to list */
1998    6                            END;
1999    5                   END;
2000    4
2001    4                   /*
2002    4                   /*      Put out node names.
2003    4                   /*/
2004    4
2005    4                   PUT_LEN = COLS_TO_DO * MFS_NOD_SEGLEN;      /* Compute length of control string */
2006    4                   FAOL_REQUESTED = YES;                       /* Run it through FAOL */
2007    4                   OUTP_REQUESTED = NO;                        /* ... but don't output it yet */
2008    4                   CALL = DISPLAY_PUT(DPUT_FLAGS,PUT_LEN,MFS_NODE_STR.S,MFS_NODE_PARMS); /* Send all nodenames to SCRPKG */
2009    4                   IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check status */
2010    4
2011    4                   /*
2012    4                   /*      Put out beginning times
2013    4                   /*/
2014    4
2015    4                   CALL SCR$SET_CURSOR(7,20);                  /* Explicitly set cursor */
2016    4                   PUT_LEN = COLS_TO_DO * MFS_TIM_SEGLEN;      /* Compute length of control string */
2017    4                   FAOL_REQUESTED = YES;                       /* Run it through FAOL */
2018    4                   OUTP_REQUESTED = NO;                        /* ... but don't output it yet */
2019    4                   CALL = DISPLAY_PUT(DPUT_FLAGS,PUT_LEN,MFS_TIME_STR.S,MFS_BEG_PARMS); /* Send all beg times to SCRPKG */
2020    4                   IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check status */
2021    4
2022    4                   /*
2023    4                   /*      Put out ending times
2024    4                   /*/
2025    4
2026    4                   CALL SCR$SET_CURSOR(8,20);                  /* Explicitly set cursor */
2027    4                   CALL = DISPLAY_PUT(DPUT_FLAGS,PUT_LEN,MFS_ IME_STR.S,MFS_END_PARMS); /* Send all end times to SCRPKG */
2028    4                                                               /* Note -- uses most arguments from previous call */
2029    4                   IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check status */
2030    4
2031    4                   END;
```

```
2032    3
2033    3
```

```
2034    3       IF STAT_IND = YES                                       /* If statistics headings requested, */
2035    3           THEN DO;
2036    4
2037    4                   /*
2038    4                   /*      Put out statistics column headings for stats (TOT AVE MIN MAX).
2039    4                   /*/
2040    4
2041    4                   STHEAD_COL = 20 + (19 * COLS_TO_DO) - 1;        /* Calculate column number */
2042    4                   IF COLS_TO_DO = 0                               /* If no data columns, */
2043    4                       THEN STHEAD_COL = STHEAD_COL + 9;           /*  then move stats columns to the right */
2044    4                   CALL SCR$SET_CURSOR(7,STHEAD_COL);              /* Explicitly set cursor */
2045    4                   PUT_LEN = MFS_STHEAD1_STR.L;                    /* Length of stats head control string */
2046    4                   FAOL_REQUESTED = YES;                           /* Run it through FAOL */
2047    4                   OUTP_REQUESTED = NO;                            /* ... but don't output it yet */
2048    4                   CALL = DISPLAY_PUT(DPUT_FLAGS,PUT_LEN,MFS_STHEAD1_STR.S,); /* Send 1st stats heading line to SCRPKG */
2049    4                   IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL);   /* Check status */
2050    4
2051    4                   CALL SCR$SET_CURSOR(8,STHEAD_COL);              /* Explicitly set cursor */
2052    4                   PUT_LEN = MFS_STHEAD2_STR.L;                    /* Length of stats head control string */
2053    4                   CALL = DISPLAY_PUT(DPUT_FLAGS,PUT_LEN,MFS_STHEAD2_STR.S,); /* Send 2nd stats heading line to SCRPKG */
2054    4                   IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL);   /* Check status */
2055    4
2056    4                   END;
2057    3
2058    3       RETURN(NORMAL);
2059    3
2060    3       END MFSUM_HEADINGS;
2061    2
2062    2       END MF_SUMMARY_EVENT;
2063    1
```

```
2064    1       MF_REQ_CLEANUP: Procedure;                              /* M.F. Summary cleanup procedure */
2065    2
2066    2       /*
2067    2       /*++
2068    2       /*
2069    2       /* FUNCTIONAL DESCRIPTION:
2070    2       /*
2071    2       /*      MF_REQ_CLEANUP
2072    2       /*
2073    2       /*      Called by MFSUM_REQUEST, either in mainline path, or
2074    2       /*      from its condition handler, to close files, free
2075    2       /*      virtual memory and generally release acquired resources.
2076    2       /*
2077    2       /* INPUTS:
2078    2       /*
2079    2       /*      None
2080    2       /*
2081    2       /* OUTPUTS:
2082    2       /*
2083    2       /*      None
2084    2       /*
2085    2       /* ROUTINE VALUE:
2086    2       /*
2087    2       /*      None
2088    2       /*
2089    2       /*--
2090    2       /*/
2091    2
2092    2       Declare
2093    2       MFS_FREE_MEM    ENTRY RETURNS(FIXED BINARY(31));        /* MONITOR MACRO-32 routine to issue LIB$FREE_VM's *
2094    2
2095    2
2096    2       IF CTRLCZ_CHAN ^= 0                                     /* If CTRL/C and CTRL/Z handlers present, */
2097    2           THEN DO;
2098    3                CALL = SYS$CANCEL(CTRLCZ_CHAN);                /* Get rid of them */
2099    3                CTRLCZ_CHAN = 0;                               /* ... and indicate their disestablishment */
2100    3                END;
2101    2
2102    2       IF M->MRB$V_INP_CL_REQ                                  /* If cleanup required for an input file, */
2103    2           THEN CALL = INPUT_CLEANUP();                        /* ... then do cleanup for it */
2104    2
2105    2       IF M->MRB$V_SUM_CL_REQ                                  /* If summary cleanup required, */
2106    2           THEN DO;
2107    3                M->MRB$A_SUMMARY = MFS$A_SUMMARY;              /* Get /SUMMARY file spec ptr to MRB for call */
2108    3                CALL = SUMMARY_CLEANUP();                      /* ... and do it */
2109    3                END;
2110    2
2111    2       CALL = MFS_FREE_MEM();                                  /* Free CDB$A_SUMBUF and CDX$A_SELIDTABLE */
2112    2       FREE MFS;                                               /* Free Multi-File Summary Block */
2113    2
2114    2       RETURN;
2115    2
2116    2       END MF_REQ_CLEANUP;
2117    1
2118    1       END MFSUM_REQUEST;
```

```
  COMMAND LINE
  ------- ----

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

H 6

M
V

HOMOG
LIS

MONITOR
LIS

MFSUMM
LIS

MONDAT
LIS