```
MMM         MMM     OOOOOOOOO     NNN         NNN  TTTTTTTTTTTTTTT     OOOOOOOOO     RRRRRRRRRRRR
MMM         MMM     OOOOOOOOO     NNN         NNN  TTTTTTTTTTTTTTT     OOOOOOOOO     RRRRRRRRRRRR
MMM         MMM     OOOOOOOOO     NNN         NNN  TTTTTTTTTTTTTTT     OOOOOOOOO     RRRRRRRRRRRR
MMMMMM   MMMMMM  OOO         OOO  NNN         NNN        TTT        OOO         OOO  RRR         RRR
MMMMMM   MMMMMM  OOO         OOO  NNN         NNN        TTT        OOO         OOO  RRR         RRR
MMMMMM   MMMMMM  OOO         OOO  NNN         NNN        TTT        OOO         OOO  RRR         RRR
MMM   MMM   MMM  OOO         OOO  NNNNNN      NNN        TTT        OOO         OOO  RRR         RRR
MMM   MMM   MMM  OOO         OOO  NNNNNN      NNN        TTT        OOO         OOO  RRR         RRR
MMM   MMM   MMM  OOO         OOO  NNNNNN      NNN        TTT        OOO         OOO  RRR         RRR
MMM         MMM  OOO         OOO  NNN   NNN   NNN        TTT        OOO         OOO  RRRRRRRRRRRR
MMM         MMM  OOO         OOO  NNN   NNN   NNN        TTT        OOO         OOO  RRRRRRRRRRRR
MMM         MMM  OOO         OOO  NNN   NNN   NNN        TTT        OOO         OOO  RRRRRRRRRRRR
MMM         MMM  OOO         OOO  NNN      NNNNNN        TTT        OOO         OOO  RRR RRR
MMM         MMM  OOO         OOO  NNN      NNNNNN        TTT        OOO         OOO  RRR RRR
MMM         MMM  OOO         OOO  NNN      NNNNNN        TTT        OOO         OOO  RRR RRR
MMM         MMM  OOO         OOO  NNN         NNN        TTT        OOO         OOO  RRR      RRR
MMM         MMM  OOO         OOO  NNN         NNN        TTT        OOO         OOO  RRR      RRR
MMM         MMM  OOO         OOO  NNN         NNN        TTT        OOO         OOO  RRR      RRR
MMM         MMM     OOOOOOOOO     NNN         NNN        TTT           OOOOOOOOO     RRR         RRR
MMM         MMM     OOOOOOOOO     NNN         NNN        TTT           OOOOOOOOO     RRR         RRR
MMM         MMM     OOOOOOOOO     NNN         NNN        TTT           OOOOOOOOO     RRR         RRR
```

```
**FILE**ID**REQUEST                                    D  6

RRRRRRRR    EEEEEEEEEE     QQQQQQ    UU      UU  EEEEEEEEEE    SSSSSSSS  TTTTTTTTTT
RRRRRRRR    EEEEEEEEEE     QQQQQQ    UU      UU  EEEEEEEEEE    SSSSSSSS  TTTTTTTTTT
RR      RR  EE            QQ    QQ   UU      UU  EE           SS             TT
RR      RR  EE            QQ    QQ   UU      UU  EE           SS             TT
RR      RR  EE            QQ    QQ   UU      UU  EE           SS             TT
RRRRRRRR    EEEEEEEE      QQ    QQ   UU      UU  EEEEEEEE      SSSSSS        TT
RRRRRRRR    EEEEEEEE      QQ    QQ   UU      UU  EEEEEEEE      SSSSSS        TT
RR RR       EE            QQ QQ QQ   UU      UU  EE                 SS       TT
RR RR       EE            QQ QQ QQ   UU      UU  EE                 SS       TT
RR   RR     EE            QQ   QQ    UU      UU  EE                 SS       TT            ....
RR   RR     EE            QQ   QQ    UU      UU  EE                 SS       TT            ....
RR      RR  EEEEEEEEEE     QQQQ QQ   UUUUUUUUUU  EEEEEEEEEE   SSSSSSSS       TT            ....
RR      RR  EEEEEEEEEE     QQQQ QQ   UUUUUUUUUU  EEEEEEEEEE   SSSSSSSS       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
```

EXE
VO4

```
EXECUTE_REQUEST: Procedure      Returns(Fixed Binary(31))
                                Options(Ident('V04-000'));

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

/*
/*++
/* FACILITY:  MONITOR Utility
/*
/* ABSTRACT:  EXECUTE_REQUEST Routine.
/*
/*              Called from MONMAIN routine to execute a single
/*                MONITOR request.
/*
/*
/* ENVIRONMENT:
/*
/*              Unprivileged user mode,
/*              except for certain collection routines which
/*              run in EXEC or KERNEL mode to access system
/*              data bases.
/*
/* AUTHOR:  Thomas L. Cafarella, April, 1981
/*
```

```
 1      /*
 1      /* MODIFIED BY:
 1      /*
 1      /*      V03-026 TLC1091         Thomas L. Cafarella     08-Aug-1984     15:00
 1      /*              Save summary buffer data for only those classes requested;
 1      /*              exclude extra classes collected in support of SYSTEM class.
 1      /*
 1      /*      V03-025 TLC1090         Thomas L. Cafarella     02-Aug-1984     15:00
 1      /*              Correct ACCVIOs in SYSTEM and PROCESSES classes.
 1      /*
 1      /*      V03-024 TLC1086         Thomas L. Cafarella     24-Jul-1984     14:00
 1      /*              Make top summary work for SYSTEM class.
 1      /*
 1      /*      V03-023 TLC1085         Thomas L. Cafarella     22-Jul-1984     14:00
 1      /*              Calculate scale values for Free and Modified List bar graphs.
 1      /*
 1      /*      V03-022 TLC1082         Thomas L. Cafarella     23-Jul-1984     11:00
 1      /*              Always save data in summary buffers, even when only
 1      /*                      one collection.
 1      /*
 1      /*      V03-021 TLC1072         Thomas L. Cafarella     17-Apr-1984     11:00
 1      /*              Add volume name to DISK display.
 1      /*
 1      /*      V03-020 TLC1068         Thomas L. Cafarella     13-Apr-1984     14:00
 1      /*              Fix bug causing a garbage heading display.
 1      /*
 1      /*      V03-019 PRS1019         Paul R. Senn            11-Apr-1984     16:00
 1      /*              Fix SYSTEM class /SUMMARY and change SYSTEM default interval.
 1      /*
 1      /*      V03-018 TLC1060         Thomas L. Cafarella     12-Mar-1984     11:00
 1      /*              Make multi-file summary work for homogeneous classes.
 1      /*
 1      /*      V03-018 TLC1059         Thomas L. Cafarella     20-Mar-1984     11:00
 1      /*              When re-recording, include input revision level
 1      /*              in output file.
 1      /*
 1      /*      V03-018 TLC1057         Thomas L. Cafarella     22-Mar-1984     15:00
 1      /*              Eliminate node name from heading for multi-file summary.
 1      /*
 1      /*      V03-018 TLC1056         Thomas L. Cafarella     22-Mar-1984     11:00
 1      /*              Disable journaling classes and exclude class which is disabled.
 1      /*
 1      /*      V03-017 PRS1011         Paul R. Senn            29-Feb-1984     14:00
 1      /*              add /FLUSH_INTERVAL qualifier
 1      /*
 1      /*      V03-016 TLC1052         Thomas L. Cafarella     17-Feb-1984     11:00
 1      /*              Add multi-file summary capability.
 1      /*
 1      /*      V03-015 TLC1051         Thomas L. Cafarella     11-Jan-1984     11:00
 1      /*              Add consecutive number to class header record.
 1      /*
 1      /*      V03-015 PRS1003         Paul R. Senn            9-Jan-1984      10:00
 1      /*              Fix 1 bit parameter passing problem on call to DISP_TEMPLATE.
 1      /*
 1      /*      V03-015 PRS1002         Paul R. Senn            29-Dec-1983     16:00
 1      /*              GLOBALDEF VALUE symbols must now be longwords;
 ¦ 1    /*              Use %REPLACE rather than GLOBALDEF VALUE for any equated
 ¦ 1    /*              symbols which are not 4 bytes in length;
 ¦ 1    /*
 ¦ 1    /*
 ¦ 1    /*      V03-015 PRS1001         Paul R. Senn            27-Dec-1983     16:00
 ¦ 1    /*              Make default interval = 6 for ALL classes Pseudo-class
 ¦ 1    /*              live requests.
 ¦ 1    /*
 ¦ 1    /*      V03-015 PRS1000         Paul R. Senn            15-Dec-1983     16:00
 ¦ 1    /*              For cases where one display event may involve multiple
 ¦ 1    /*              screens of data (such as PROCESSES and homogeneous
 ¦ 1    /*              classes), make the wait between screens = VIEWING_TIME,
 ¦ 1    /*              instead of a constant of 2 seconds.
 ¦ 1    /*
 ¦ 1    /*      V03-014 TLC1050         Thomas L. Cafarella     06-Dec-1983     11:00
 ¦ 1    /*              Change directory information in DLOCK class.
 ¦ 1    /*
 ¦ 1    /*      V03-013 TLC1047         Thomas L. Cafarella     09-Sep-1983     10:00
 ¦ 1    /*              De-establish CTRL/W handler to get back AST quota.
 ¦ 1    /*
 ¦ 1    /*      V03-012 SPC0007         Stephen P. Carney       24-Jun-1983     16:00
 ¦ 1    /*              Add execute command file handling in CTRLZ routine.
 ¦ 1    /*
 ¦ 1    /*      V03-011 TLC1042         Thomas L. Cafarella     19-Jun-1983     15:00
 ¦ 1    /*              Add /ITEM qualifier for homogeneous classes.
 ¦ 1    /*
 ¦ 1    /*      V03-011 TLC1039         Thomas L. Cafarella     15-Jun-1983     15:00
 ¦ 1    /*              Add DECnet node name to heading.
 ¦ 1    /*
 ¦ 1    /*      V03-011 TLC1037         Thomas L. Cafarella     14-Jun-1983     19:00
 ¦ 1    /*              Perform FLUSH after writing record (instead of before).
 ¦ 1    /*
 ¦ 1    /*      V03-011 SPC0005         Stephen P. Carney       10-Jun-1983     15:00
 ¦ 1    /*              Make the playback/record file read-shareable
 ¦ 1    /*
 ¦ 1    /*      V03-010 TLC1035         Thomas L. Cafarella     06-Jun-1983     15:00
 ¦ 1    /*              Add homogeneous class type and DISK class.
 ¦ 1    /*
 ¦ 1    /*      V03-010 TLC1033         Thomas L. Cafarella     30-May-1983     16:00
 ¦ 1    /*              Don't clear screen for CTRL/Z.
 ¦ 1    /*
 ¦ 1    /*      V03-009 TLC1032         Thomas L. Cafarella     27-May-1983     15:00
 ¦ 1    /*              Modify file structure level ID for LOCK class change.
 ¦ 1    /*
 ¦ 1    /*      V03-008 SPC0002         Stephen P. Carney       22-Apr-1983     14:00
 ¦ 1    /*              Modify file structure level ID for new ACPCACHE class.
 ¦ 1    /*
 ¦ 1    /*      V03-007 TLC1028         Thomas L. Cafarella     14-Apr-1983     16:00
 ¦ 1    /*              Add interactive user interface.
 ¦ 1    /*
 ¦ 1    /*      V03-007 TLC1027         Thomas L. Cafarella     14-Apr-1983     16:00
 ¦ 1    /*              Enhance file compatibility features.
 ¦ 1    /*
 ¦ 1    /*      V03-006 TLC1022         Thomas L. Cafarella     12-Jul-1982     16:00
 ¦ 1    /*              Change recording file structure level since new classes
 ¦ 1    /*              (JOURNALING and RECOVERY) are now defined.
 ¦ 1    /*
 1      /*      V03-005 TLC1016         Thomas L. Cafarella     02-Apr-1982     16:00
 1      /*              Replace references to EXE$GQ_SYSTIME with $GETTIM calls.
 1      /*
 1      /*      V03-005 TLC1014         Thomas L. Cafarella     01-Apr-1982     13:00
 1      /*              Correct attached processor time reporting for MODES class.
 1      /*
 1      /*      V03-005 TLC1013         Thomas L. Cafarella     31-Mar-1982     09:00
 1      /*              Do not clear TOP box until it fills with data.
 1      /*
 1      /*      V03-005 TLC1012         Thomas L. Cafarella     30-Mar-1982     13:00
 1      /*              Display user's comment string on screen line 5.
 1      /*
 1      /*      V03-005 TLC1011         Thomas L. Cafarella     29-Mar-1982     20:00
 1      /*              Move system service names for SSERROR msg to static storage.
 1      /*
 1      /*      V03-004 TLC1009         Thomas L. Cafarella     29-Mar-1982     01:00
 1      /*              Get current time when other times are converted.
 1      /*
 1      /*      V03-004 TLC1008         Thomas L. Cafarella     28-Mar-1982     21:00
 1      /*              Fix to display first and last PROCESSES records on playback.
 1      /*
 1      /*      V03-004 TLC1006         Thomas L. Cafarella     28-Mar-1982     13:00
 1      /*              Add checks to skip data display on CTRL-C during template.
 1      /*
 1      /*      V03-003 TLC1003         Thomas L. Cafarella     23-Mar-1982     13:00
 1      /*              Fix up module headers.
 1      /*
 1      /*      V03-002 TLC1002         Thomas L. Cafarella     20-Mar-1982     13:00
 1      /*              Change PROCESSES display from scroll-style to page-style to
 1      /*                      make it terminal-independent.
 1      /*
 1      /*              Move collection event flag to REQUEST.PLI for consolidation.
 1      /*
 1      /*      V03-001 TLC1001         Thomas L. Cafarella     16-Mar-1982     13:00
 1      /*              Add CTRL-W screen refresh support.
 1      /*
 1      /*--
 1      /*/
 1
 | 1       /*
 | 1       /*      |-------------------------------------------------------------+
 | 1       /*      |                                                             |
 | 1       /*      |                        INCLUDE  FILES                       |
 | 1       /*      |                                                             |
 | 1       /*      |-------------------------------------------------------------+
 | 1       /*/
   1
   1       %INCLUDE          MONDEF;                  /* Monitor utility structure definitions */
   1       %INCLUDE          $CHFDEF;                 /* Condition handler facility definitions */
   1       %INCLUDE          $STSDEF;                 /* Status value definitions */
   1
 | 1       /*
 | 1       /*      |-------------------------------------------------------------+
 | 1       /*      |                                                             |
 | 1       /*      |              SYSTEM SERVICE MACRO DEFINITIONS               |
 | 1       /*      |                                                             |
 | 1       /*      |-------------------------------------------------------------+
 | 1       /*/
   1
   1       %INCLUDE          SYS$DCLAST;              /* $DCLAST system service */
   1       %INCLUDE          SYS$SETAST;              /* $SETAST system service */
   1       %INCLUDE          SYS$CLREF;               /* $CLREF system service */
   1       %INCLUDE          SYS$SETEF;               /* $SETEF system service */
   1       %INCLUDE          SYS$READEF;              /* $READEF system service */
   1       %INCLUDE          SYS$SETIMR;              /* $SETIMR system service */
   1       %INCLUDE          SYS$CANTIM;              /* $CANTIM system service */
   1       %INCLUDE          SYS$ASCTIM;              /* $ASCTIM system service */
   1       %INCLUDE          SYS$WAITFR;              /* $WAITFR system service */
   1       %INCLUDE          SYS$WFLOR;               /* $WFLOR system service */
   1       %INCLUDE          SYS$PUTMSG;              /* $PUTMSG system service */
   1
 ¦   1      /*
 ¦   1      /*      |------------------------------------------------------------------+
 ¦   1      /*      |                                                                  |
 ¦   1      /*      |                 EXTERNAL STORAGE  DEFINITIONS                    |
 ¦   1      /*      |                                                                  |
 ¦   1      /*      |------------------------------------------------------------------+
 ¦   1      /**/
     1
     1      Declare
     1      ST_LEVEL_CUR     CHAR(8)              GLOBALREF;              /* Current MONITOR recording file structure level */
     1
     1      Declare
     1      MAX_CLASS_NO     FIXED BINARY(31) GLOBALREF VALUE,            /* Maximum defined class number */
     1      CLASSTABLE       FIXED BINARY(31) GLOBALREF VALUE,            /* Addr of table of class names & numbers*/
     1      VTWIDTH          FIXED BINARY(31) GLOBALREF VALUE,            /* Width of video terminal */
     1      VTHEIGHT         FIXED BINARY(31) GLOBALREF VALUE,            /* Height of video terminal */
     1      MAX_REC_SIZE     FIXED BINARY(31) GLOBALREF VALUE,            /* Max record size for PLAYBACK & RECORD files */
     1      PROCS_CLSNO      FIXED BINARY(31) GLOBALREF VALUE,            /* PROCESSES class number */
     1      STATES_CLSNO     FIXED BINARY(31) GLOBALREF VALUE,            /* STATES class number */
     1      MODES_CLSNO      FIXED BINARY(31) GLOBALREF VALUE,            /* MODES class number */
     1      SYSTEM_CLSNO     FIXED BINARY(31) GLOBALREF VALUE;            /* SYSTEM class number */
     1
     1      Declare
     1      CDBPTR                   POINTER GLOBALREF,                   /* Pointer to CDB (Class Descriptor Block) */
     1      C                        POINTER DEFINED(CDBPTR),             /* Synonym for CDBPTR */
     1      MRBPTR                   POINTER GLOBALREF,                   /* Pointer to MRB (Monitor Request Block) */
     1      M                        POINTER DEFINED(MRBPTR),             /* Synonym for MRBPTR */
     1      MCAPTR                   POINTER GLOBALREF,                   /* Pointer to MCA (Monitor Communication Area) */
     1      MC                       POINTER DEFINED(MCAPTR),             /* Synonym for MCAPTR */
     1      SPTR                     POINTER GLOBALREF;                   /* Pointer to SYI (System Information Area) */
     1
     1      Declare
     1      MFSPTR                   POINTER GLOBALREF;                   /* Pointer to Multi-File Summary Block (MFS) */
     1
     1      Declare
     1      DISPLAYING               BIT(1) ALIGNED GLOBALREF;            /* YES=> display output is active */
     1
     1      Declare
     1      CTRLCZ_CHAN              FIXED BINARY(31) GLOBALREF,          /* Channel number for CTRL-C and CTRL-Z */
     1      CTRLW_CHAN               FIXED BINARY(31) GLOBALREF;          /* Channel number for CTRL-W */
     1
     1      Declare
     1      EXE$GL_MP                POINTER GLOBALREF,                   /* Pointer to multiprocessing data structures */
     1      SGN$GW_MAXPRCCT          FIXED BINARY(15) GLOBALREF,          /* MAXPROCESSCNT SYSGEN parameter value */
     1      PFN$GL_PHYPGCNT          FIXED BINARY(31) GLOBALREF,          /* Balance set memory size (in pages) */
     1      MPW$GW_HILIM             FIXED BINARY(15) GLOBALREF;          /* MPW_HILIMIT SYSGEN parameter value */
     1
     1      Declare
     1      SETIMR_STR               FIXED BINARY(7)  GLOBALREF,          /* Count byte for $SETIMR cstring */
     1      DCLAST_STR               FIXED BINARY(7)  GLOBALREF,          /* Count byte for $DCLAST cstring */
     1      SCHDWK_STR               FIXED BINARY(7)  GLOBALREF,          /* Count byte for $SCHDWK cstring */
     1      READEF_STR               FIXED BINARY(7)  GLOBALREF,          /* Count byte for $READEF cstring */
     1      CLREF_STR                FIXED BINARY(7)  GLOBALREF;          /* Count byte for $CLREF cstring */
     1
     1
 |  1       /*
 |  1       /*        |----------------------------------------------------------------------+
 |  1       /*        |                                                                      |
 |  1       /*        |                 EXTERNAL ROUTINE DEFINITIONS                         |
 |  1       /*        |                                                                      |
 |  1       /*        |----------------------------------------------------------------------+
 |  1       /*/
    1
    1       Declare
    1       MON_ERR           ENTRY (ANY VALUE, ANY, ANY) OPTIONS(VARIABLE),  /* MONITOR MACRO-32 routine to log synchronous error
    1       SIGNAL_MON_ERR    ENTRY,                                           /* MONITOR MACRO-32 routine to signal MONITOR errors
    1       READ_INPUT        ENTRY (FIXED BINARY(31)),                        /* MONITOR routine to read an input (playback) file
    1       ESTAB_CTRLCZ      ENTRY RETURNS(FIXED BINARY(31)),                 /* MONITOR MACRO-32 routine to set up CTRL-C and CTR
    1       ESTAB_CTRLW       ENTRY RETURNS(FIXED BINARY(31)),                 /* MONITOR MACRO-32 routine to set up CTRL-W handler
    1       DISPLAY_INIT      ENTRY RETURNS(FIXED BINARY(31)),                 /* MONITOR MACRO-32 routine to do display init */
    1       SUMMARY_INIT      ENTRY RETURNS(FIXED BINARY(31)),                 /* MONITOR MACRO-32 routine to do summary init */
    1       COLLECTION_EVENT  ENTRY,                                           /* MONITOR routine to perform collection */
    1       QUAD_LT_QUAD      ENTRY (BIT(64) ALIGNED, BIT(64) ALIGNED)         /* MONITOR MACRO-32 unsigned quadword compare routin
    1                         RETURNS(BIT(1)),
    1       QUAD_EQ_0         ENTRY (BIT(64) ALIGNED) RETURNS(BIT(1)),         /* MONITOR MACRO-32 quadword compare to 0 routine */
    1       CVT_TO_DELTA      ENTRY (FIXED BINARY(31), BIT(64) ALIGNED),       /* MONITOR MACRO-32 rtn to convert to delta time */
    1       MPCHECK           ENTRY RETURNS(BIT(1)),                           /* MONITOR MACRO-32 rtn to check for MP capability *
    1       COMPUTE_BOOTTIME  ENTRY RETURNS(FIXED BINARY(31)),                 /* MONITOR MACRO-32 rtn to compute system time at bo
    1       CLUS_NET_INFO     ENTRY RETURNS(FIXED BINARY(31));                 /* MONITOR MACRO-32 rtn to get cluster & net info */
    1
    1       Declare
    1       DISP_TEMPLATE     ENTRY (POINTER, BIT(1) ALIGNED)                  /* Rtn to display the template */
    1                         RETURNS (FIXED BINARY(31)),
    1       DISPLAY_PUT       ENTRY(ANY, FIXED BINARY(31), ANY, ANY)           /* MACRO-32 rtn to put a display string */
    1                         OPTIONS(VARIABLE)
    1                         RETURNS(FIXED BINARY(31)),
    1       FILL_DISP_BUFF    ENTRY (POINTER, BIT(64) ALIGNED)                 /* MACRO-32 Fill display buffer routine */
    1                         RETURNS (FIXED BINARY(31)),
    1       DISPLAY_HOMOG     ENTRY (POINTER)                                  /* MACRO-32 rtn to display homog class data */
    1                         RETURNS (FIXED BINARY(31)),
    1       DISPLAY_PROCS     ENTRY (POINTER, BIT(64) ALIGNED)                 /* MACRO-32 rtn to display processes */
    1                         RETURNS (FIXED BINARY(31)),
    1       DISPLAY_TOP       ENTRY (POINTER)                                  /* MACRO-32 rtn to display top processes */
    1                         RETURNS (FIXED BINARY(31));
    1
```

```
¦  1        /*
¦  1        /*          |-----------------------------------------------------------------------+
¦  1        /*          |                                                                       |
¦  1        /*          |                       MESSAGE DEFINITIONS                             |
¦  1        /*          |                                                                       |
¦  1        /*          |-----------------------------------------------------------------------+
¦  1        /*/
   1
   1        Declare
   1        MNR$_SSERROR     FIXED BINARY(31) GLOBALREF VALUE,
   1        MNR$_DISPERR     FIXED BINARY(31) GLOBALREF VALUE,
   1        MNR$_BEGNLEND    FIXED BINARY(31) GLOBALREF VALUE,
   1        MNR$_HIB         FIXED BINARY(31) GLOBALREF VALUE,
   1        MNR$_CLASNP      FIXED BINARY(31) GLOBALREF VALUE,
   1        MNR$_CLASUNK     FIXED BINARY(31) GLOBALREF VALUE,
   1        MNR$_NOCLASS     FIXED BINARY(31) GLOBALREF VALUE,
   1        MNR$_CLASDISAB   FIXED BINARY(31) GLOBALREF VALUE,
   1        MNR$_BEGRAN      FIXED BINARY(31) GLOBALREF VALUE,
   1        MNR$_PREMEOF     FIXED BINARY(31) GLOBALREF VALUE,
   1        MNR$_INVINPFIL   FIXED BINARY(31) GLOBALREF VALUE,
   1        MNR$_UNSTLEV     FIXED BINARY(31) GLOBALREF VALUE,
   1        MNR$_NOOUTPUT    FIXED BINARY(31) GLOBALREF VALUE,
   1        MNR$_UNEXPERR    FIXED BINARY(31) GLOBALREF VALUE;
   1
   1
```

```
| 1        /*
| 1        /*        |-----------------------------------------------------------------------+
| 1        /*        |                                                                       |
| 1        /*        |                     GLOBAL STORAGE  DEFINITIONS                       |
| 1        /*        |                                                                       |
| 1        /*        |-----------------------------------------------------------------------+
| 1        /*/
  1
  1        Declare
  1        HEADER_TYPE      FIXED BINARY(15) GLOBALDEF INIT(128),            /* Type code for MONITOR recording file header */
  1        SYI_TYPE         FIXED BINARY(15) GLOBALDEF INIT(129),            /* Type code for MONITOR recording file sys info rec
  1        DISP_EV_FLAG     FIXED BINARY(31) GLOBALDEF VALUE INIT(16),       /* Display event flag */
  1        DISP_EV_FLAG_M   BIT(32) ALIGNED GLOBALDEF VALUE INIT('0000000000000001'B), /* Display event flag mask */
  1        REFR_EV_FLAG     FIXED BINARY(31) GLOBALDEF VALUE INIT(17),       /* Refresh event flag */
  1        REFR_EV_FLAG_M   BIT(32) ALIGNED GLOBALDEF VALUE INIT('00000000000000001'B), /* Refresh event flag mask */
  1        COLL_EV_FLAG     FIXED BINARY(31) GLOBALDEF VALUE INIT(18),       /* Collection event flag */
  1        BET_EV_FLAG      FIXED BINARY(31) GLOBALDEF VALUE INIT(19),       /* "Between screens" event flag */
  1        HIB_EV_FLAG      FIXED BINARY(31) GLOBALDEF VALUE INIT(20),       /* Hibernation event flag */
  1        INTERVAL_DEFAULT FIXED BINARY(31) GLOBALDEF VALUE INIT(3),        /* Default collection interval value */
  1        ALLCL_INT_DEFAULT FIXED BINARY(31) GLOBALDEF VALUE INIT(6),       /* Default coll. interval for ALL classes Pseudo cla
  1        SYSCL_INT_DEFAULT FIXED BINARY(31) GLOBALDEF VALUE INIT(6),       /* Default coll. interval for SYSTEM class Pseudo cl
  1        VIEWING_DEFAULT  FIXED BINARY(31) GLOBALDEF VALUE INIT(3),        /* Default viewing time value */
  1        FLUSH_INT_DEFAULT FIXED BINARY(31) GLOBALDEF VALUE INIT(300),     /* Default flush interval value */
  1        BALSETMEM_DEF    FIXED BINARY(31) GLOBALDEF VALUE INIT(3000),     /* Default value for balance set memory */
  1        MPWHILIM_DEF     FIXED BINARY(31) GLOBALDEF VALUE INIT(500),      /* Default value for MPW_HILIMIT */
  1        COLLENDED        BIT(1) ALIGNED   GLOBALDEF,                      /* YES => collection has ended */
  1        CTRLZ_HIT        BIT(1) ALIGNED   GLOBALDEF,                      /* YES => CTRL-Z has been hit */
  1        CTRLCZ_HIT       BIT(1) ALIGNED   GLOBALDEF,                      /* YES => CTRL-C or CTRL-Z has been hit */
  1        NEXT_REC         FIXED BINARY(31) GLOBALDEF VALUE INIT(0),        /* Read next record indicator for READ_INPUT rtn */
  1        SKIP_TO_CLASS    FIXED BINARY(31) GLOBALDEF VALUE INIT(1),        /* Skip to class record indicator for READ_INPUT rtn
  1        CCDPTR           POINTER          GLOBALDEF,                      /* Pointer to CCD (Current Class Descriptor) Array *
  1        COLL_STATUS      FIXED BINARY(31) GLOBALDEF,                      /* COLLECTION_EVENT status code */
  1        H                POINTER          GLOBALDEF,                      /* Pointer to input file header */
  1        INP_COMM_STR     CHAR(MNR_HDR$K_MAXCOMLEN) GLOBALDEF,             /* User comment string from input file */
  1        INP_COMM_LEN     FIXED BINARY(15) GLOBALDEF;                      /* Actual length of comment string */
  1
  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         ENABLE_AST                 BY 1;                 /* Enable AST indicator */
  1        %REPLACE         DISABLE_AST                BY 0;                 /* Disable AST indicator */
  1        %REPLACE         AND_OP                     BY '0001'B;           /* AND Boolean operation */
  1        %REPLACE         XOR_OP                     BY '0110'B;           /* XOR Boolean operation */
  1
```

```
        /*
        /*      |--------------------------------------------------------------+
        /*      |                                                              |
        /*      |                         OWN STORAGE                          |
        /*      |                                                              |
        /*      |--------------------------------------------------------------+
        /**/

        Declare
        CALL             FIXED BINARY(31),                            /* Holds function value (return status) of called ro
        STATUS           BIT(1)  BASED(ADDR(CALL));                   /* Low-order status bit for called routines */

        Declare
        NORMAL           FIXED BINARY(31) GLOBALREF,                  /* MONITOR normal return status */
        TEMP             FIXED BINARY(31),                            /* Scratch "register" */
        CURR_ERRCODE     FIXED BINARY(31),                            /* MONITOR error status code currently expected */
        REQUEST_STATUS   FIXED BINARY(31),                            /* EXECUTE_REQUEST status code */
        ALREADY_FAILED   BIT(1) ALIGNED,                              /* YES => a failure has already been signaled */
        TEMP_PTR         POINTER,                                     /* Scratch pointer */
        RECORD_STR       CHAR(128) VARYING;                           /* Fully expanded file name string for the recording
                                                                      /* NOTE -- When the recording file is re-opened for
                                                                      /* it uses this fully expanded file string.  This pr
                                                                      /* MONITOR from updating the wrong file if a new ver
                                                                      /* created in the directory before the recording fil
                                                                      /* opened for update. */

        Declare
        INTERVAL_DEL     BIT(64) ALIGNED GLOBALDEF,                   /* Delta time value for Interval */
        VIEWING_DEL      BIT(64) ALIGNED GLOBALDEF;                   /* Delta time value for Viewing time */

        Declare
        CURR_DCLASS      FIXED BINARY(15),                            /* Consec no (not class no) of current display class
        REPT_TOP         BIT(1) ALIGNED,                              /* YES => Repeat a TOP display */
        MULT_TEMP        FIXED BINARY(31) GLOBALDEF;                  /* Temp area for MCA$L_INT_MULT, used in COLLECTION_

        /*
        /*      File Declarations
        /*
        /*               The recording file is read shareable. This allows other
        /*               MONITOR images to use the FLUSHED recording file as input.
        /*
        /*               When the recording file is opened for update (to write header
        /*               information), it uses a fully expanded file string.  This prevents
        /*               MONITOR from updating the wrong file if a new version is created
        /*               in the directory before the recording file is opened for update.
        /**/

        Declare
        INPUT_FILE       FILE RECORD INPUT,                           /* Monitor Input (Playback) File */
        RECORD_FILE      FILE RECORD,                                 /* Monitor Record File */
        RECCT            FIXED BINARY(31) GLOBALDEF;                  /* Count of records written to record file */

        Declare
        TEMP_TYPE        BIT(8) ALIGNED;                              /* Temporary area for record type byte */

        Declare
```

```
      1         INPUT_CPTR        POINTER GLOBALDEF,                                     /* Ptr to input buffer count word */
      1         INPUT_DATA        CHAR(MAX_REC_SIZE) VARYING BASED(INPUT_CPTR),          /* Playback file input buffer */
      1         1 DPUT_FLAGS,                                                             /* DISPLAY_PUT routine flags */
      1           2 FAOL_REQUESTED BIT(8) ALIGNED,                                        /* YES => Xlate buffer with FAOL first */
      1           2 OUTP_REQUESTED BIT(8) ALIGNED,                                        /* YES => Really output buffer */
      1         PUT_LEN           FIXED BINARY(31);                                       /* Length of buffer for DISPLAY_PUT to put */
      1
      1
```

```
   1        Declare
   1        FLUSH_IND        BIT(1) ALIGNED              GLOBALDEF,                    /* Flush indicator; YES=> perform FLUSH */
   1        FLUSH_COLLS      FIXED BINARY(15)            GLOBALDEF,                    /* Number of collection events between FLUSH's */
   1        FLUSH_CTR        FIXED BINARY(15)            GLOBALDEF;                    /* Down counter for FLUSH_COLLS */
   1
   1        Declare
   1        01 CURR_CLASS_DESCR (MAX_CLASS_NO+1),                                      /* Current Class Descriptor Array */
 : 1                                                                                   /* This array of structures includes a CCD (Current
 : 1                                                                                   /* Class Descriptor) for each class collected. */
   1            02 CURR_CDBPTR       POINTER,                                          /* CDBPTR for current class */
   1            02 CURR_CLASS_NO     FIXED BINARY(7);                                  /* Class number for current class */
   1
   1
   1        Declare
   1        01 D_CURR_CLASS_DESCR (MAX_CLASS_NO+1),                                    /* Current Class Descriptor Array for display classe
 : 1                                                                                   /* This array of structures includes a D_CCD (Curren
 : 1                                                                                   /* Class Descriptor) for each class displayed. */
   1            02 D_CURR_CDBPTR     POINTER,                                          /* CDBPTR for current display class */
   1            02 D_CURR_CLASS_NO   FIXED BINARY(7);                                  /* Class number for current display class */
   1
 1      /*
 1      /*++
 1      /*
 1      /* FUNCTIONAL DESCRIPTION:
 1      /*
 1      /*      EXECUTE_REQUEST
 1      /*
 1      /*      This routine is called by MONMAIN to execute a MONITOR request.
 1      /*      The request is defined by the Monitor Request Block (MRB), which
 1      /*      is created by MONMAIN after parsing a command string.
 1      /*
 1      /* INPUTS:
 1      /*
 1      /*      None
 1      /*
 1      /* IMPLICIT INPUTS:
 1      /*
 1      /*      Monitor Request Block (MRB), pointed to by MRBPTR.
 1      /*
 1      /* OUTPUTS:
 1      /*
 1      /*      None
 1      /*
 1      /* IMPLICIT OUTPUTS:
 1      /*
 1      /*      Monitor request has been performed.
 1      /*
 1      /* ROUTINE VALUE:
 1      /*
 1      /*      SS$_NORMAL, or failing MONITOR status code.
 1      /*
 1      /*--
 1      /*/
 1
```

```
     1        ON FINISH;                                                         /* On finish, do nothing */
     1        ALREADY_FAILED = NO;                                               /* Indicate no failure yet signaled */
     1        CURR_ERRCODE = 0;                                                  /* Set expected MONITOR code to default */
     1
 !   1        /*
 !   1        /*      Set up condition handler to terminate the MONITOR request on:
 !   1        /*              1) any asynchronous error condition, such as file and I/O errors;
 !   1        /*              2) any synchronous MONITOR-detected condition.
 !   1        /*/
     1
     1        ON ANYCONDITION                                                    /* On any condition signaled, */
     1        BEGIN;
     2
     2        Declare
     2          MNR$_ERRINPFIL          FIXED BINARY(31) GLOBALREF VALUE,       /* Error message code */
     2          MNR$_ERRRECFIL          FIXED BINARY(31) GLOBALREF VALUE,       /* Error message code */
     2          MNR$_UNEXPERR           FIXED BINARY(31) GLOBALREF VALUE,       /* Error message code */
     2          MON_CODE                FIXED BINARY(31),                       /* Monitor message code */
     2          TEMP                    FIXED BINARY(31),                       /* Temporary scratch area */
     2          MNR$_FACNO              FIXED BINARY(31) GLOBALREF VALUE,       /* MONITOR facility number */
     2          ON_FILE                 CHAR(100) VARYING,                      /* Holds possible file name string */
     2          SIGNALED_ERR ENTRY (ANY VALUE, ANY VALUE, ANY VALUE, ANY);      /* Rtn to set up PUTMSGVEC */
     2
     2        IF ^ ALREADY_FAILED                                                /* If a failure not already signaled, */
     2            THEN DO;
     3                ALREADY_FAILED = YES;                                      /* Indicate a failure has been signaled */
     3                CHF$ARGPTR = ONARGSLIST();                                 /* Get signal array pointer */
     3                STS$VALUE = CHF$SIG_NAME;                                  /* Get code for signaled condition */
     3                UNSPEC(TEMP) = STS$FAC_NO;                                 /* Convert facility no. to binary in TEMP */
     3                IF TEMP = MNR$_FACNO                                       /* If a MONITOR code, */
     3                    THEN MON_CODE = STS$VALUE;                             /*      then remember it */
     3                ELSE DO;                                                   /* Otherwise, need to set the MON_CODE */
     4                    ON_FILE = ONFILE();                                        /* Get PL/I file constant if I/O cond */
     4                    IF ON_FILE = 'INPUT_FILE'                                  /* If input file error, */
     4                        THEN MON_CODE = MNR$_ERRINPFIL;                        /* Set Monitor status code accordingly */
     4                        ELSE IF ON_FILE = 'RECORD_FILE'                        /* See if it's the recording file */
     4                                THEN MON_CODE = MNR$_ERRRECFIL;                /* Yes, save code */
     4                                ELSE IF CURR_ERRCODE = 0                       /* No, see if an error is currently expected
     4                                        THEN MON_CODE = MNR$_UNEXPERR;         /* No, set "unexpected" code */
     4                                        ELSE MON_CODE = CURR_ERRCODE;          /* Yes, set currently expected code */
     4                    CURR_ERRCODE = 0;                                          /* Reset to default MONITOR error code ("une
     4                    CALL SIGNALED_ERR(MON_CODE,STS$VALUE,DIM(CHF$SIG_ARG,1),CHF$SIG_ARG); /* Log the error */
     4                    END;
     3
     3                REQUEST_STATUS = MON_CODE;                                 /* Set up code for MONITOR request termination */
     3                CALL = COLLECTION_END();                                   /* Shut down collection activity */
     3                CALL REQUEST_CLEANUP();                                    /* Perform cleanup for files, memory, etc. */
     3                END;
     2
     2        GO TO REQUEST_EXIT;                                                /* Go return from EXECUTE_REQUEST (PL/I does an UNWI
     2
     2        END;                                                               /* End of ON-condition routine */
     1
```

```
        /*
        /*      Set up EOF condition
        /*/

        IF M->MRB$A_INPUT ^= NULL()                                /* If this is a PLAYBACK request, */
            THEN  ON ENDFILE(INPUT_FILE) MC->MCA$V_EOF = YES;      /* then set up EOF condition */

        /*
        /*      General MONITOR request initialization
        /*/

        CALL = REQUEST_INIT();                                     /* Initialization for this request */
        IF STATUS = NOT_SUCCESSFUL
            THEN CALL SIGNAL_MON_ERR();                            /* Short-circuit request if failure */

        /*
        /*      Establish CTRL-C and CTRL-Z handlers for terminating the MONITOR request.
        /*      CTRL-C causes a MONITOR> prompt. CTRL-Z returns to DCL.
        /*/

        IF COLLENDED = NO  THEN CALL = ESTAB_CTRLCZ();             /* If still collecting, establish CTRL-C and CTRL-Z handlers
                                                                   /* If error, do not terminate; simply ignore CTRL-C's and CT

        /*
        /*      Establish CTRL-W handler for refreshing the terminal display.
        /*/

        IF COLLENDED = NO & M->MRB$V_DISPLAY                       /* If still collecting, and display requested */
            THEN CALL = ESTAB_CTRLW();                             /* ... establish CTRL-W handler */
                                                                   /* If error, do not terminate; simply ignore CTRL-W's */
```

```
 ! 1         /*
 ! 1         /*      If this is a live request, beginning in the future, "hibernate"
 ! 1         /*      the process until ready to execute. (Use event flags instead of $HIBER
 ! 1         /*      to avoid the problem of outstanding wakeups interfering with later
 ! 1         /*      MONITOR requests.)
 ! 1         /**/
   1
   1         IF ^ M->MRB$V_PLAYBACK & MC->MCA$V_FUTURE                /* If live future request, */
   1                              & COLLENDED = NO                    /* ... and not terminated, */
   1             THEN DO;
   2                  CALL = SYS$SETIMR(HIB_EV_FLAG,M->MRB$Q_BEGINNING,,HIB_EV_FLAG);
 ! 2                                                                   /* ... set flag when ready to execute request */
   2                  IF STATUS = NOT_SUCCESSFUL                       /* Failed? */
   2                      THEN DO;
   3                           CALL MON_ERR(MNR$_SSERROR,CALL,SETIMR_STR); /* Yes -- log the error */
   3                           CALL SIGNAL_MON_ERR();                  /* ... and signal it */
   3                           END;
   2                  BEGIN;
   3                  DECLARE 1 HIBMSG,                                /* Declare hibernate msg vec dynamically */
   3                           2 HCOUNT FIXED BIN(31) INIT(1),
   3                           2 HMSG   FIXED BIN(31) INIT(MNR$_HIB);
   3                  CALL = SYS$PUTMSG(HIBMSG,,);                     /* Let user know we're sleeping */
   3                  END;
   2                  IF COLLENDED = NO  THEN CALL = SYS$WAITFR(HIB_EV_FLAG); /* ... ZZZZZZZZZZ */
   2                  END;
   1
 ! 1         /*
 ! 1         /*      Initialization associated with /DISPLAY output
 ! 1         /**/
   1
   1         IF M->MRB$V_DISPLAY                                       /* If DISPLAY has been requested, */
   1            & COLLENDED = NO                                       /* ... and request not terminated, */
   1             THEN DO;
   2                  CALL = DISPLAY_INIT();                           /* ... then perform init for it */
   2                  IF STATUS = NOT_SUCCESSFUL                       /* Failed? */
   2                      THEN DO;
   3                           CALL MON_ERR(MNR$_DISPERR,CALL);        /* Yes -- log the error */
   3                           CALL SIGNAL_MON_ERR();                  /* ... and signal it */
   3                           END;
   2                  ON FINISH CALL = DISPLAY_CLEANUP();              /* On finish, do display cleanup */
   2                  END;
   1
 ! 1         /*
 ! 1         /*      Initialization associated with /RECORD output
 ! 1         /**/
   1
   1         IF M->MRB$V_RECORD                                        /* If RECORD has been requested, */
   1            & COLLENDED = NO                                       /* ... and request not terminated, */
   1             THEN DO;
   2                  CALL = RECORD_INIT();                            /* ... then do init for it */
   2                  IF STATUS = NOT_SUCCESSFUL
   2                      THEN CALL SIGNAL_MON_ERR();                  /* Signal error if failure */
   2                  END;
   1
 ! 1        /*
 ! 1        /*      Execute main monitoring routine. When control returns from the CALL,
 ! 1        /*      the MONITOR request will have terminated.
 ! 1        /*/
   1
   1        IF COLLENDED = NO                                          /* If collection not ended, */
   1            THEN DO;
   2                 CALL = MONITOR_REQUEST();                         /* Perform the MONITOR request */
   2                 IF STATUS = NOT_SUCCESSFUL                        /* If failed, */
   2                     THEN CALL SIGNAL_MON_ERR();                   /* ... signal the error */
   2                 END;
   1
 ! 1        /*
 ! 1        /*      Perform SUMMARY processing if requested.
 ! 1        /*/
   1
   1        IF M->MRB$V_SUMMARY                                        /* If SUMMARY has been requested, */
   1            THEN DO;
   2                 CALL = REQUEST_SUMMARY();                         /* Perform SUMMARY processing */
   2                 IF STATUS = NOT_SUCCESSFUL                        /* If failed, */
   2                     THEN CALL SIGNAL_MON_ERR();                   /* ... signal the error */
   2                 END;
   1
 ! 1        /*
 ! 1        /*      Perform Multi-File Summary processing if requested.
 ! 1        /*/
   1
   1        IF M->MRB$V_MFSUM                                          /* If Multi-File Summary has been requested, */
   1            THEN DO;
   2                 CALL = SAVE_SUM_BUFFS();                          /* Save SUM buffers */
   2                 IF STATUS = NOT_SUCCESSFUL                        /* Failed? */
   2                     THEN DO;
   3                          CALL MON_ERR(MNR$_UNEXPERR,CALL);        /* Yes -- log the error */
   3                          CALL SIGNAL_MON_ERR();                   /* ... and signal it */
   3                          END;
   2                 END;
   1
 ! 1        /*
 ! 1        /*      Cleanup processing
 ! 1        /*/
   1
   1        CALL REQUEST_CLEANUP();                                    /* Execute various cleanup routines */
   1
 ! 1        /*
 ! 1        /*      Exit from EXECUTE_REQUEST routine
 ! 1        /*      Note -- we get to this point either by falling through
 ! 1        /*      the above code (normal path), or by direct branch from
 ! 1        /*      the condition-handling routine (error path).
 ! 1        /*/
   1
   1        REQUEST_STATUS = NORMAL;                                   /* Normal status if we get to this point */
   1
   1        REQUEST_EXIT:
   1
   1        RETURN(REQUEST_STATUS);                                    /* Return to MONMAIN.PLI with status */
   1
 1       REQUEST_INIT: Procedure Returns(fixed binary(31));
 2
 2 |     /*
 2 |     /*++
 2 |     /*
 2 |     /* FUNCTIONAL DESCRIPTION:
 2 |     /*
 2 |     /*       REQUEST_INIT
 2 |     /*
 2 |     /*       Performs initialization for the Monitor request.
 2 |     /*       Fills in defaults for the MRB (Monitor Request Block).
 2 |     /*       Also inits the MCA (Monitor Communication Area), opens
 2 |     /*       the input (playback) file if necessary, and fills in the
 2 |     /*       SYI (System Information Area).
 2 |     /*
 2 |     /* INPUTS:
 2 |     /*
 2 |     /*       None
 2 |     /*
 2 |     /* OUTPUTS:
 2 |     /*
 2 |     /*       None
 2 |     /*
 2 |     /* ROUTINE VALUE:
 2 |     /*
 2 |     /*       SS$_NORMAL, or failing MONITOR status code.
 2 |     /*
 2 |     /* SIDE EFFECTS:
 2 |     /*
 2 |     /*       /INPUT file (INPUT_FILE) is positioned to the first class record.
 2 |     /*
 2 |     /*--
 2 |     /*/
 2
 2 |     /*
 2 |     /*       +--------------------------------------------------------------------------+
 2 |     /*       |                                                                          |
 2 |     /*       |                              LOCAL STORAGE                               |
 2 |     /*       |                                                                          |
 2 |     /*       +--------------------------------------------------------------------------+
 2 |     /*/
 2
 2       Declare
 2         F             POINTER;                                  /* Pointer to class record in input file */
 2
```

```
   2      REQUEST_STATUS = NORMAL;                                  /* Start off this MONITOR request with normal status */
   2      COLL_STATUS = NORMAL;                                     /* Start off COLLECTION_EVENT with normal status */
   2      COLLENDED = NO;                                           /* Indicate collection has not ended */
   2      CTRLZ_HIT = NO;                                           /* Indicate CTRL-Z not hit */
   2      CTRLCZ_HIT = NO;                                          /* Indicate CTRL-C and CTRL-Z not hit */
   2      CTRLCZ_CHAN = 0;                                          /* ... and no channel assigned for them */
   2      CTRLW_CHAN = 0;                                           /* No channel assigned for CTRL-W */
   2      INP_COMM_LEN = 0;                                         /* Length of input comment string */
   2      MC->MCA$L_COLLCNT = 0;                                    /* Initialize collection count */
   2      MC->MCA$L_DISPCNT = 0;                                    /* ... and display count */
   2      MC->MCA$L_CONSEC_REC = 0;                                 /* Consecutive collection number for recording */
   2      MC->MCA$Q_LASTCOLL = '0'B;                                /* Init latest collection time */
   2      MC->MCA$V_FUTURE = NO;                                    /* Assume not a future request */
   2      MC->MCA$V_EOF = NO;                                       /* Assume EOF not yet found on /INPUT file */
   2      MC->MCA$V_ERA_SCRL = NO;                                  /* Assume no need to erase scrolling region */
   2      MC->MCA$V_VIDEO = NO;                                     /* Assume display terminal is not video */
   2      MC->MCA$V_GRAPHICS = NO;                                  /* ... and not VT-55 graphics */
   2      MC->MCA$V_TOP_DISP = NO;                                  /* Indicate no TOP displays issued yet */
   2      MC->MCA$V_S_TOP_DISP = NO;                                /* Also no SYSTEM (TOP) displays issued yet */
   2      MC->MCA$V_REFRESH = NO;                                   /* Indicate screen refresh request not received */
   2      MC->MCA$V_FILLER = '0'B;                                  /* Clear all unused flags */
   2      FLUSH_IND = NO;                                           /* Indicate recording file flush not required */
   2      CURR_DCLASS = 0;                                          /* Init current display class */
   2      REPT_TOP = NO;                                            /* Indicate do not repeat TOP display */
   2      DISPLAYING = NO;                                          /* Indicate display output not yet begun */
   2      CCDPTR = ADDR(CURR_CLASS_DESCR);                          /* Set up ptr for COLLECTION_EVENT to use */
   2      CALL = SYS$CLREF(REFR_EV_FLAG);                           /* Clear refresh event flag */
   2      CALL = SYS$SETAST(ENABLE_AST);                            /* Make sure AST's are enabled */
   2
 ! 2      /*
 ! 2      /*      Set MRB flags for options that were requested
 ! 2      /*/
   2
   2      IF M->MRB$A_INPUT   ^= NULL() THEN M->MRB$V_PLAYBACK = YES; /* If INPUT specified, indicate so */
   2      IF M->MRB$A_DISPLAY ^= NULL() THEN M->MRB$V_DISPLAY = YES;  /* If DISPLAY specified, indicate so */
   2      IF M->MRB$A_RECORD  ^= NULL() THEN M->MRB$V_RECORD = YES;   /* If RECORD specified, indicate so */
   2      IF M->MRB$A_SUMMARY ^= NULL() THEN M->MRB$V_SUMMARY = YES;  /* If SUMMARY specified, indicate so */
   2
   2      IF ^ M->MRB$V_DISPLAY & ^ M->MRB$V_RECORD & ^ M->MRB$V_SUMMARY /* If none of the outputs requested, */
   2       & ^ M->MRB$V_MFSUM                                       /* ... AND it's not a m.f. summary, */
   2          THEN DO;
   3                CALL MON_ERR(MNR$_NOOUTPUT);                    /* Log the error */
   3                RETURN(MNR$_NOOUTPUT);                          /* ... and return with status */
   3                END;
   2
 ! 2      /*
 ! 2      /*      Set or clear display event flag
 ! 2      /*/
   2
   2      IF M->MRB$V_DISPLAY                                       /* If display requested, */
   2          THEN CALL = SYS$SETEF(DISP_EV_FLAG);                  /* ... then set display event flag to force 1st display even
   2          ELSE CALL = SYS$CLREF(DISP_EV_FLAG);                  /* ... otherwise clear it */
   2
 ! 2      /*
 ! 2      /*      If PLAYBACK, perform input file initialization, so MONITOR file header information can be accessed.
 ! 2      /*/
```

```
    2
    2       IF M->MRB$V_PLAYBACK                                        /* If this is a PLAYBACK request, */
    2           THEN DO;
    3                CALL = INPUT_INIT();                               /* ... perform input file initializaton */
    3                IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL);
    3                END;
    2
```

```
     2        IF M->MRB$V_PLAYBACK                                            /* If this is a PLAYBACK request, */
     2            THEN DO;
     3                 INP_COMM_STR = H->MNR_HDR$T_COMMENT;                   /* Save user's comment string from header record */
     3                 INP_COMM_LEN = H->MNR_HDR$W_COMLEN;                    /* ... and its length */
     3                 END;
     2
 !   2        /*
 !   2        /*        The next several groups of code update the MRB with default and
 !   2        /*        specified values, and, for PLAYBACK, values from the input file.
 !   2        /*/
     2
 !   2        /*
 !   2        /*        Verify requested classes and set up Current Class Descriptor array
 !   2        /*/
     2
     2        BEGIN;
     3        Declare
     3        SELECT_REV_LEVS ENTRY(BIT(128) ALIGNED, BIT(128) ALIGNED, CHAR(128), ANY) /* MACRO-32 rtn to select revision levels
/
     3                        OPTIONS(VARIABLE),                                     /* ... for all classes */
     3        CALC_LEN        ENTRY(BIT(128) ALIGNED)                                /* MACRO-32 rtn to calculate class block (
     3                        RETURNS (FIXED BINARY(31));
     3        Declare
     3        REVLEVELS       (0:127) FIXED BINARY(7) GLOBALDEF,              /* Revision Levels Vector */
     3        REVOCLSBITS     BIT(128) GLOBALDEF,                             /* Monitored classes still at Rev Level 0 */
     3        REVOCB_VEC      (0:127) BIT(1) DEFINED(REVOCLSBITS);            /* Bit-addressable alias */
     3        Declare
     3          FILE_CLASSES  BIT(128),                                       /* Classes from input file */
     3          REQ_CLASSES   BIT(128),                                       /* Classes requested by user */
     3          NP_CLASSES    BIT(MAX_CLASS_NO+1),                            /* Classes requested but not in input file */
     3          DO_CLASSES    BIT(128),                                       /* Classes to actually monitor */
     3          DO_CLASSES_VEC (0:127) BIT(1) DEFINED(DO_CLASSES),            /* Bit-addressable alias */
     3          DO_CLASSES_AL BIT(128) ALIGNED,                               /* Aligned copy of DO_CLASSES */
     3          UNK_CLASSES   BIT(128) ALIGNED,                               /* Classes with unknown revision levels */
     3          DISPLAY_CLASSES BIT(128) ALIGNED GLOBALDEF,                   /* Classes to be displayed */
     3          CDBHEAD       FIXED BINARY(31) GLOBALREF VALUE,               /* Address of first CDB */
     3          I             FIXED BINARY(15),                               /* Index for do-loop */
     3          CLASS_NO      FIXED BINARY(7),                                /* Class number */
     3          TEMP_CDBPTR   POINTER,                                        /* Ptr to Class Descriptor Block (CDB) */
     3          CDBPTR_COMP   FIXED BINARY(31) BASED(ADDR(TEMP_CDBPTR));      /* Computable CDBPTR */
     3
     3        Declare
     3          SYS_REQ       BIT(1) ALIGNED,                                 /* YES => SYSTEM class requested */
     3          PROCS_REQ     BIT(1) ALIGNED,                                 /* YES => PROCESSES class requested */
     3          STATES_REQ    BIT(1) ALIGNED,                                 /* YES => STATES class requested */
     3          MODES_REQ     BIT(1) ALIGNED;                                 /* YES => MODES class requested */
     3
```

```
        3         DO I = 0 TO 127;                                                       /* Set all revision levels ... */
        4         REVLEVELS(I) = 0;                                                      /* ... to 0 */
        4         END;
        3
        3         DO_CLASSES = M->MRB$O_CLASSBITS;                                       /* Get set of classes to do */
        3
        3         IF M->MRB$V_PLAYBACK                                                   /* Playback request */
        3             THEN DO;
        4                  IF MNR_HDR$K_CLASSBITS < MC->MCA$L_INPUT_LEN                  /* If CLASSBITS field is defined for input file, */
        4                      THEN FILE_CLASSES = H->MNR_HDR$O_CLASSBITS;               /*   then get file classes from usual place */
        4                      ELSE FILE_CLASSES = H->MNR_HDR$O_REVOCLSBITS;             /*   else get them from another place */
  |     4                                                                                /* NOTE -- MNR_HDR$O_REVOCLSBITS is used for compati
  |     4                                                                                /*          with MONSL001 and MONBA001 file struct le
        4                  REQ_CLASSES = DO_CLASSES;                                     /* Get requested classes */
        4                  DO_CLASSES = BOOL(FILE_CLASSES,REQ_CLASSES,AND_OP);           /* Compute classes to actually do */
        4                  NP_CLASSES = BOOL(DO_CLASSES,REQ_CLASSES,XOR_OP);             /* Compute classes not present */
        4                  IF DO_CLASSES = '0'B                                          /* If no classes to be done, */
        4                      THEN DO;
        5                           CALL MON_ERR(MNR$_NOCLASS);                          /* Log error */
        5                           RETURN(MNR$_NOCLASS);                                /* ... and return */
        5                           END;
        4
        4                  IF M->MRB$V_DISPLAY = NO & NP_CLASSES ^= '0'B                 /* If at least one class not present AND not display
        4                     & M->MRB$V_MFSUM = NO & M->MRB$V_ALL_CLASS = NO            /* ... AND not multi-file summary, AND not ALL_CLASS
        4                      THEN BEGIN;                                               /* ... then print a warning */
        5                           DECLARE 1 NPMSG,                                     /* Declare not present msg vec dynamically */
        5                                     2 NPCOUNT FIXED BIN(31) INIT(1),
        5                                     2 NPMSG   FIXED BIN(31) INIT(MNR$_CLASNP);
        5                           CALL = SYS$PUTMSG(NPMSG,,);                          /* Warn user that classes missing */
        5                           END;
        4                  END;
        3
  3       IF DO_CLASSES_VEC(SYSTEM_CLSNO)                                  /* If SYSTEM class requested, */
  3           THEN DO;
  4               SYS_REQ = YES;                                           /* Remember that fact */
  4               PROCS_REQ = DO_CLASSES_VEC(PROCS_CLSNO);                 /* Remember whether PROCESSES requested */
  4               DO_CLASSES_VEC(PROCS_CLSNO) = YES;                       /* ... and include it */
  4               STATES_REQ = DO_CLASSES_VEC(STATES_CLSNO);               /* Remember whether STATES requested */
  4               DO_CLASSES_VEC(STATES_CLSNO) = YES;                      /* ... and include it */
  4               MODES_REQ = DO_CLASSES_VEC(MODES_CLSNO);                 /* Remember whether MODES requested */
  4               DO_CLASSES_VEC(MODES_CLSNO) = YES;                       /* ... and include it */
  4               END;
  3           ELSE SYS_REQ = NO;                                           /* Indicate SYSTEM class not requested */
  3
  3       DO_CLASSES_AL = DO_CLASSES;                                      /* Get aligned string for later routine calls */
  3
  3       IF M->MRB$V_PLAYBACK                                             /* Playback request */
  3           THEN DO;
! 4               /*
! 4               /*      For each class in DO_CLASSES, update the CDB with information
! 4               /*      from the CHD (CHange Descriptor) for the appropriate revision
! 4               /*      level.
! 4               /*
! 4               /*      Eliminate from DO_CLASSES those classes with incompatible structure
! 4               /*      levels. Issue a warning message if any incompatibilities found.
! 4               /*/
  4
  4               IF MNR_HDR$K_REVLEVELS < MC->MCA$L_INPUT_LEN             /* If REVLEVELS field is defined for input file, */
  4                   THEN DO;
  5                       CALL SELECT_REV_LEVS(DO_CLASSES_AL, UNK_CLASSES,
  5                                   H->MNR_HDR$T_REVLEVELS, REVLEVELS); /* Select revision levels ... */
! 5                                                                        /* ... for all classes */
  5                       IF  UNK_CLASSES ^= '0'B                          /* If at least one class has unknown rev level, */
  5                           THEN DO;
  6                               DO_CLASSES = BOOL(DO_CLASSES,UNK_CLASSES,XOR_OP); /* Remove unknowns from DO_CLASSES */
  6                               IF M->MRB$V_DISPLAY = NO                 /* If not displaying, */
  6                                   THEN BEGIN;                          /* ... then print a warning */
  7                                       DECLARE 1 UNKMSG,                /* Declare unknown msg vec dynamically */
  7                                               2 UNKCOUNT FIXED BIN(31) INIT(1),
  7                                               2 UNKMSG   FIXED BIN(31) INIT(MNR$_CLASUNK);
  7                                       CALL = SYS$PUTMSG(UNKMSG,,); /* Warn user that classes have unknown structs */
  7                                       END;
  6                               END;
  5                       END;
  4
  4                   ELSE                                                 /* Revision levels all 0 */
  4                       CALL SELECT_REV_LEVS(DO_CLASSES_AL,,,REVLEVELS); /* Move CHD info into CDB */
! 4                                                                        /* ... for all classes */
  4               END;
  3
  3           ELSE DO;                                                     /* Live request */
  4               CALL SELECT_REV_LEVS(DO_CLASSES_AL,,,REVLEVELS);         /* Select revision levels for all classes */
  4               END;
  3
  3       IF DO_CLASSES_VEC(SYSTEM_CLSNO) = YES                            /* If SYSTEM class being monitored, */
  3           THEN M->MRB$V_SYSCLS = YES;                                  /*   then indicate so */
  3           ELSE DO;
  4               M->MRB$V_SYSCLS = NO;                                    /*   else indicate not */
```

```
4                  IF SYS_REQ = YES                                          /* If SYSTEM originally requested, */
4                      THEN DO;                                              /*    then make some further checks */
5                           IF PROCS_REQ = NO                                /* If PROCESSES not originally requested, */
5                               THEN DO_CLASSES_VEC(PROCS_CLSNO) = NO;       /* ... make sure it's off now */
5                           IF STATES_REQ = NO                               /* If STATES not originally requested, */
5                               THEN DO_CLASSES_VEC(STATES_CLSNO) = NO;      /* ... make sure it's off now */
5                           IF MODES_REQ = NO                                /* If MODES not originally requested, */
5                               THEN DO_CLASSES_VEC(MODES_CLSNO) = NO;       /* ... make sure it's off now */
5                           END;
4                  END;
3
3         IF M->MRB$V_MFSUM                                                  /* If multi-file summary, */
3             THEN DO_CLASSES_VEC(PROCS_CLSNO) = NO;                         /*    then make sure PROCESSES class is still off */
3
3         IF DO_CLASSES = '0'B                                               /* If no classes to be done, */
3             THEN DO;
4                  CALL MON_ERR(MNR$_NOCLASS);                               /* Log error */
4                  RETURN(MNR$_NOCLASS);                                     /* ... and return */
4                  END;
3
3         DISPLAY_CLASSES = M->MRB$O_CLASSBITS;                              /* Get unaligned copy of orig requested classes */
3         DISPLAY_CLASSES = BOOL(DISPLAY_CLASSES,DO_CLASSES,AND_OP);         /* Compute classes to display */
3         M->MRB$O_CLASSBITS = DO_CLASSES;                                   /* Remember classes to collect */
3
```

```
 !  3        /*
 !  3        /*      Given DO_CLASSES, execute do loop using INDEX builtin
 !  3        /*      to fill in the CCD (Current Class Descriptor) array.
 !  3        /*      When do loop is finished, MRB$O_CLASSBITS, MRB$W_CLASSCT
 !  3        /*                              MCA$B_FIRSTC and MCA$B_LASTC
 !  3        /*                              will all be established.
 !  3        /*/
    3
    3        REV0CLSBITS = '0'B;                                             /* Assume no classes at Rev Level 0 */
    3        CALL = CALC_LEN(M->MRB$O_CLASSBITS);                            /* Calc CDB$W_BLKLEN field for each class */
    3        IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL);                   /* Return if error */
    3
    3        CLASS_NO = 0;                                                   /* Initialize class number */
    3        DO I = 1 TO MAX_CLASS_NO + 1  WHILE(CLASS_NO >= 0);             /* Loop once for each possible class */
    4          CLASS_NO = INDEX(DO_CLASSES,YES) - 1;                         /* Find next requested class number */
    4          IF CLASS_NO >= 0                                              /* Only continue if a class was found */
    4             THEN DO;
    5                  DO_CLASSES_VEC(CLASS_NO) = NO;                        /* Eliminate it from future consideration */
    5                  IF REVLEVELS(CLASS_NO) = 0                            /* If this class is at Rev Level 0, */
    5                     THEN REV0CB_VEC(CLASS_NO) = YES;                   /*   then indicate so */
    5                  CURR_CLASS_NO(I) = CLASS_NO;                          /* Store class_no in CCD table */
    5                  IF I = 1 THEN MC->MCA$B_FIRSTC = CLASS_NO;            /* Mark first class requested */
    5                  MC->MCA$B_LASTC = CLASS_NO;                           /* Mark last class requested */
    5                  M->MRB$W_CLASSCT = I;                                 /* Keep track of class count */
    5                  CDBPTR_COMP = CDBHEAD + (CDB$K_SIZE * CURR_CLASS_NO(I));
 !  5                                                                        /* Compute current cdbptr ... */
    5                  CURR_CDBPTR(I) = TEMP_CDBPTR;                         /* ... and store it in CCD table */
    5                  END;
    4        END;
    3
 !  3        /*
 !  3        /*      Now, given DISPLAY_CLASSES, do a similar loop as above to set up D_CCD,
 !  3        /*      the display version of the CCD. When loop is finished, the array will
 !  3        /*      be established along with MCA$W_DCLASSCT, the number of display classes.
 !  3        /*/
    3
    3        DO_CLASSES = DISPLAY_CLASSES;                                   /* Use DO_CLASSES vector */
    3        IF DO_CLASSES_VEC(PROCS_CLSNO)                                  /* If PROCESSES to be displayed, */
    3           THEN M->MRB$V_PROC_REQ = YES;                                /*   then indicate it was requested */
    3           ELSE M->MRB$V_PROC_REQ = NO;                                 /*   else indicate not */
    3        CLASS_NO = 0;                                                   /* Initialize class number */
    3        DO I = 1 TO MAX_CLASS_NO + 1  WHILE(CLASS_NO >= 0);             /* Loop once for each possible class */
    4          CLASS_NO = INDEX(DO_CLASSES,YES) - 1;                         /* Find next requested class number */
    4          IF CLASS_NO >= 0                                              /* Only continue if a class was found */
    4             THEN DO;
    5                  DO_CLASSES_VEC(CLASS_NO) = NO;                        /* Eliminate it from future consideration */
    5                  D_CURR_CLASS_NO(I) = CLASS_NO;                        /* Store class_no in D_CCD table */
    5                  MC->MCA$W_DCLASSCT = I;                               /* Keep track of class count */
    5                  CDBPTR_COMP = CDBHEAD + (CDB$K_SIZE * D_CURR_CLASS_NO(I));
 !  5                                                                        /* Compute current cdbptr .... */
    5                  D_CURR_CDBPTR(I) = TEMP_CDBPTR;                       /* ... and store it in D_CCD table */
    5                  END;
    4        END;
    3
    3        END;                                                            /* End of BEGIN-END group */
    2
 | 2      /*
 | 2      /*      Establish defaults for FLUSH_INTERVAL, INTERVAL and VIEWING_TIME
 | 2      /*      options.
 | 2      /*      If Playback, divide file value for INTERVAL into requested value, and round
 | 2      /*      requested value up to the next whole multiple of the file value. Store the
 | 2      /*      multiple value in MCA$L_INT_MULT. It will be used to trigger recording and
 | 2      /*      display events.
 | 2      /*/
   2
   2      IF M->MRB$L_FLUSH = 0                                                /* If FLUSH never specified... */
   2          THEN M->MRB$L_FLUSH = FLUSH_INT_DEFAULT;                         /* normal default value */
   2
   2      IF M->MRB$V_PLAYBACK                                                 /* Playback request */
   2          THEN DO;
   3              IF M->MRB$L_VIEWING_TIME = 0                                 /* If VIEWING_TIME never specified, */
   3                  THEN M->MRB$L_VIEWING_TIME = VIEWING_DEFAULT;            /* ... then take default */
   3              IF M->MRB$L_INTERVAL = 0                                     /* If INTERVAL never specified, */
   3                  THEN DO;
   4                      M->MRB$L_INTERVAL = H->MNR_HDR$L_INTERVAL;           /* ... then use file value */
   4                      MC->MCA$L_INT_MULT = 1;                              /* ... and multiple of 1 */
   4                      END;
   3                  ELSE DO;                                                 /* INTERVAL explicitly specified */
   4                      MC->MCA$L_INT_MULT = DIVIDE(M->MRB$L_INTERVAL,H->MNR_HDR$L_INTERVAL,31);
 | 4                                                                           /* Divide spec'd val by file val */
   4                      IF (M->MRB$L_INTERVAL - (H->MNR_HDR$L_INTERVAL * MC->MCA$L_INT_MULT)) ^= 0
   4                          THEN DO;
   5                              MC->MCA$L_INT_MULT = MC->MCA$L_INT_MULT + 1; /* Round up if necessary */
   5                              M->MRB$L_INTERVAL = MC->MCA$L_INT_MULT * H->MNR_HDR$L_INTERVAL; /* Round interval too */
   5                              END;
   4                      END;
   3              END;
   2
   2          ELSE DO;                                                         /* Live request */
   3              IF M->MRB$L_INTERVAL = 0                                     /* If INTERVAL never specified... */
   3                  THEN IF M->MRB$V_ALL_CLASS                               /* ALL class request */
   3                          THEN M->MRB$L_INTERVAL = ALLCL_INT_DEFAULT;      /* ALL class default value */
   3                          ELSE IF M->MRB$V_SYSCLS                          /* SYSTEM class request */
   3                                  THEN M->MRB$L_INTERVAL = SYSCL_INT_DEFAULT;  /* SYSTEM class default value */
   3                                  ELSE M->MRB$L_INTERVAL = INTERVAL_DEFAULT;      /* normal default value */
   3              IF M->MRB$L_VIEWING_TIME = 0                                 /* If VIEWING_TIME never specified... */
   3                  THEN M->MRB$L_VIEWING_TIME = M->MRB$L_INTERVAL;          /* Default to interval value */
   3              IF M->MRB$L_INTERVAL <= M->MRB$L_FLUSH                       /* Requested interval not larger than flush period?
   3                  THEN FLUSH_COLLS = DIVIDE(M->MRB$L_FLUSH,M->MRB$L_INTERVAL,31); /* Yes -- compute collections until flu
   3                  ELSE FLUSH_COLLS = 1;                                    /* No -- flush on every collection */
   3              FLUSH_CTR = FLUSH_COLLS;                                     /* Set up down counter */
   3
   3              END;
   2
   2      CALL CVT_TO_DELTA(M->MRB$L_INTERVAL,INTERVAL_DEL);                   /* Convert INTERVAL to delta time */
   2      CALL CVT_TO_DELTA(M->MRB$L_VIEWING_TIME,VIEWING_DEL);                /* Convert VIEWING_TIME to delta time */
   2
 :  2       /*
 :  2       /*      Establish defaults for BEGINNING and ENDING options
 :  2       /**/
    2
    2       IF M->MRB$V_PLAYBACK
    2           THEN MC->MCA$Q_CURR_TIME = H->MNR_HDR$Q_BEGINNING;             /* If Playback, get current time from file */
 :  2                                                                          /* If Live, MCA$Q_CURR_TIME already contains */
 :  2                                                                          /* ... current time from system */
    2
 :  2       /*
 :  2       /*      If user requested a past time for the BEGINNING option,
 :  2       /*      or defaulted, then replace his value with MCA$Q_CURR_TIME.
 :  2       /*      Otherwise, indicate a future request.
 :  2       /**/
    2
    2       MC->MCA$V_FUTURE = QUAD_LT_QUAD(MC->MCA$Q_CURR_TIME,M->MRB$Q_BEGINNING); /* MCA$V_FUTURE gets YES or NO */
    2       IF MC->MCA$V_FUTURE = NO
    2           THEN M->MRB$Q_BEGINNING = MC->MCA$Q_CURR_TIME;                 /* If NO, give user current time */
    2
 :  2       /*
 :  2       /*      For PLAYBACK, verify ENDING option. If file value is
 :  2       /*      non-zero, replace requested value with file value if
 :  2       /*      requested value is 0 (never specified), or requested
 :  2       /*      value is larger (later) than file value.
 :  2       /**/
    2
    2       IF M->MRB$V_PLAYBACK
    2           THEN IF ^ QUAD_EQ_0(H->MNR_HDR$Q_ENDING)
    2                   THEN IF QUAD_EQ_0(M->MRB$Q_ENDING)
    2                           THEN M->MRB$Q_ENDING = H->MNR_HDR$Q_ENDING;
    2                           ELSE IF QUAD_LT_QUAD(H->MNR_HDR$Q_ENDING,M->MRB$Q_ENDING)
    2                                   THEN M->MRB$Q_ENDING = H->MNR_HDR$Q_ENDING;
    2
 :  2       /*
 :  2       /*      Set indefinite end bit if ENDING option never specified.
 :  2       /*
 :  2       /*      Also, perform sanity check of BEGINNING and ENDING times.
 :  2       /*      If BEGINNING not less than ENDING, exit with error.
 :  2       /**/
    2
    2       IF QUAD_EQ_0(M->MRB$Q_ENDING)                                      /* If ENDING never specified, */
    2           THEN M->MRB$V_INDEFEND = YES;                                  /* ... call it indefinite */
    2           ELSE IF QUAD_LT_QUAD(M->MRB$Q_BEGINNING,M->MRB$Q_ENDING) = NO  /* If BEGINNING not less than ENDING, */
    2               THEN DO;
    3                   CALL MON_ERR(MNR$_BEGNLEND);                           /* Log the error */
    3                   RETURN(MNR$_BEGNLEND);                                 /* ... and return with status */
    3                   END;
    2
 ! 2        /*
 ! 2        /*      Get information about the monitored system.
 ! 2        /*/
   2
   2        IF M->MRB$V_PLAYBACK                                      /* PLAYBACK request */
   2           THEN DO;
   3                CALL READ_INPUT(NEXT_REC);                        /* Read system information record */
   3                IF MC->MCA$V_EOF                                  /* If end-of-file, */
   3                   THEN DO;
   4                        CALL MON_ERR(MNR$_PREMEOF);               /* Can't find sys info record; log the error */
   4                        RETURN (MNR$_PREMEOF);                    /* ... and return to caller */
   4                        END;
   3
   3                TEMP_PTR = MC->MCA$A_INPUT_PTR;                   /* Establish ptr to sys info record */
   3                TEMP_TYPE = UNSPEC(SYI_TYPE);                     /* Get sys info type into a byte for compare */
   3                IF TEMP_PTR->MNR_SYI$B_TYPE ^= TEMP_TYPE          /* If this record is not the sys info rec, */
   3                   THEN DO;
   4                        CALL MON_ERR(MNR$_INVINPFIL);             /* Log an error */
   4                        RETURN(MNR$_INVINPFIL);                   /* ... and return to caller */
   4                        END;
   3
 ! 3                /*
 ! 3                /* Move entire sys info record to System Information Area
 ! 3                /*/
   3
   3                SPTR->MNR_SYI$B_TYPE = TEMP_PTR->MNR_SYI$B_TYPE;          /* Get SYI type code */
   3                SPTR->MNR_SYI$W_FLAGS = TEMP_PTR->MNR_SYI$W_FLAGS;        /* Get all flags */
   3                SPTR->MNR_SYI$B_MPCPUS = TEMP_PTR->MNR_SYI$B_MPCPUS;      /* Get number of cpu's */
   3                SPTR->MNR_SYI$W_MAXPRCCT = TEMP_PTR->MNR_SYI$W_MAXPRCCT; /* Get MAXPROCESSCNT SYSGEN parameter */
   3                SPTR->MNR_SYI$Q_BOOTTIME = TEMP_PTR->MNR_SYI$Q_BOOTTIME; /* Get system boot time */
   3
   3                IF MNR_SYI$K_NODENAME < MC->MCA$L_INPUT_LEN               /* If NODENAME field is defined for input file, */
   3                   THEN SPTR->MNR_SYI$T_NODENAME = TEMP_PTR->MNR_SYI$T_NODENAME; /* ... then pick it up from there */
   3                   ELSE UNSPEC(SPTR->MNR_SYI$T_NODENAME) = '0'B;          /* Otherwise, simply clear it */
   3
   3                IF MNR_SYI$K_BALSETMEM < MC->MCA$L_INPUT_LEN              /* If BALSETMEM field is defined for input file, */
   3                   THEN SPTR->MNR_SYI$L_BALSETMEM = TEMP_PTR->MNR_SYI$L_BALSETMEM; /* ... then pick it up from there */
   3                   ELSE SPTR->MNR_SYI$L_BALSETMEM = BALSETMEM_DEF;        /* Otherwise, use a constant default value */
   3
   3                IF MNR_SYI$K_MPWHILIM < MC->MCA$L_INPUT_LEN               /* If MPWHILIM field is defined for input file, */
   3                   THEN SPTR->MNR_SYI$L_MPWHILIM = TEMP_PTR->MNR_SYI$L_MPWHILIM; /* ... then pick it up from there */
   3                   ELSE SPTR->MNR_SYI$L_MPWHILIM = MPWHILIM_DEF;          /* Otherwise, use a constant default value */
   3
   3                IF MNR_SYI$K_CPUTYPE < MC->MCA$L_INPUT_LEN                /* If CPUTYPE field is defined for input file, */
   3                   THEN SPTR->MNR_SYI$L_CPUTYPE = TEMP_PTR->MNR_SYI$L_CPUTYPE; /* ... then pick it up from there */
   3                   ELSE SPTR->MNR_SYI$L_CPUTYPE = 0;                      /* Otherwise, simply clear */
   3
   3                END;
   2
```

```
    2         ELSE DO;                                            /* LIVE request */
  ! 3                                                             /* Fill the System Information Area from the running system
    3              SPTR->MNR_SYI$B_TYPE = UNSPEC(SYI_TYPE);       /* Get SYI type code */
    3              SPTR->MNR_SYI$V_RESERVED1 = '0'B;              /* Clear reserved flag ... */
    3              SPTR->MNR_SYI$V_FILLER = '0'B;                 /* ... and all unused flags */
    3              IF MPCHECK()                                   /* Multiprocessing capability? */
    3                  THEN SPTR->MNR_SYI$B_MPCPUS = 2;           /* Yes -- 2 cpu's */
    3                  ELSE SPTR->MNR_SYI$B_MPCPUS = 1;           /* No -- just 1 cpu */
    3              SPTR->MNR_SYI$W_MAXPRCCT = SGN$GW_MAXPRCCT;    /* Get MAXPROCESSCNT SYSGEN parameter */
    3              CALL = COMPUTE_BOOTTIME();                     /* Get system time at boot into MNR_SYI$Q_BOOTTIME */
    3              IF STATUS = NOT_SUCCESSFUL                     /* Failed? */
    3                  THEN DO;
    4                       CALL MON_ERR(MNR$_UNEXPERR,CALL);     /* Yes -- log the error */
    4                       RETURN(MNR$_UNEXPERR);                /* ... and return with status */
    4                       END;
    3
    3              CALL = CLUS_NET_INFO();                        /* Get cluster and network info (incl CPU type) into SYI */
    3              IF STATUS = NOT_SUCCESSFUL                     /* Failed? */
    3                  THEN DO;
    4                       CALL MON_ERR(MNR$_UNEXPERR,CALL);     /* Yes -- log the error */
    4                       RETURN(MNR$_UNEXPERR);                /* ... and return with status */
    4                       END;
    3
    3              SPTR->MNR_SYI$L_BALSETMEM = PFN$GL_PHYPGCNT;   /* Get balance set memory size (in pages) */
    3              SPTR->MNR_SYI$L_MPWHILIM = MPW$GW_HILIM;       /* Get MPW_HILIMIT SYSGEN parameter */
    3
    3              END;
    2
```

```
        /*
        /*      If PLAYBACK, read first class record from input file
        /*      to "prime the pump."
        /*/

        IF M->MRB$V_PLAYBACK
            THEN DO;
                CALL READ_INPUT(SKIP_TO_CLASS);                 /* Set up first class record for COLLECTION_EVENT rtn */
                IF MC->MCA$V_EOF                                /* If end-of-file, */
                    THEN DO;
                        CALL MON_ERR(MNR$_PREMEOF);             /* Log the error */
                        RETURN (MNR$_PREMEOF);                  /* ... and return to caller */
                        END;

        /*
        /*      If a future playback request, read input file, skipping
        /*      past class records until file is positioned to requested
        /*      begin point. Examine file time value only for the first
        /*      class record within an interval, to ensure that the request
        /*      will begin at an interval boundary. If end-of-file is hit
        /*      during this operation, terminate the request with an error.
        /*/

                IF MC->MCA$V_FUTURE
                    THEN DO;
                        F = MC->MCA$A_INPUT_PTR;
                        DO WHILE (^ MC->MCA$V_EOF & QUAD_LT_QUAD(F->MNR_CLS$Q_STAMP,M->MRB$Q_BEGINNING));

                        READ FILE(INPUT_FILE) INTO(INPUT_DATA); /* Read rec following first class record */

                        DO WHILE (^ MC->MCA$V_EOF & F->MNR_CLS$B_TYPE ^= MC->MCA$B_FIRSTC);

                        READ FILE(INPUT_FILE) INTO(INPUT_DATA); /* Read until first class found again */

                        END;
                        END;
                        IF MC->MCA$V_EOF                        /* EOF => bad beginning time */
                            THEN DO;
                                CALL MON_ERR(MNR$_BEGRAN);      /* Log the error */
                                RETURN(MNR$_BEGRAN);            /* ... and return with status */
                                END;
                        END;

                MC->MCA$L_INPUT_LEN = LENGTH(INPUT_DATA);       /* Establish length of input */
                END;

        RETURN(NORMAL);                                         /* Return to caller */
        END REQUEST_INIT;
```

```
     1       RECORD_INIT: Procedure Returns(fixed binary(31));
     2
  |  2       /*
  |  2       /*++
  |  2       /*
  |  2       /* FUNCTIONAL DESCRIPTION:
  |  2       /*
  |  2       /*      RECORD_INIT
  |  2       /*
  |  2       /*      Called by EXECUTE_REQUEST to open the output (recording) file.
  |  2       /*
  |  2       /* INPUTS:
  |  2       /*
  |  2       /*      None
  |  2       /*
  |  2       /* OUTPUTS:
  |  2       /*
  |  2       /*      None
  |  2       /*
  |  2       /* ROUTINE VALUE:
  |  2       /*
  |  2       /*      SS$_NORMAL
  |  2       /*
  |  2       /*--
  |  2       /*/
     2
  |  2       /*
  |  2       /*      |-------------------------------------------------------------+
  |  2       /*      |                                                             |
  |  2       /*      |                        LOCAL STORAGE                        |
  |  2       /*      |                                                             |
  |  2       /*      |-------------------------------------------------------------+
  |  2       /*/
     2
     2       %INCLUDE PLI_FILE_DISPLAY;
     2
     2       Declare
     2         RECORD_EXPTR  POINTER,
     2         TEMP_PTR      POINTER,
     2         01 TEMP BASED(TEMP_PTR),
     2           02 L        FIXED BINARY(15),
     2           02 DC       FIXED BINARY(15),
     2           02 A        POINTER,
     2         TEMP_STR      CHAR(TEMP.L) BASED(TEMP.A);
     2
     2       RECCT = 0;                                                 /* Init count of records written */
     2       M->MRB$V_REC_CL_REQ = YES;                                 /* Indicate record cleanup is required */
     2       CLOSE FILE(RECORD_FILE);                                   /* Make sure file is closed before opening */
     2       TEMP_PTR = M->MRB$A_RECORD;                                /* Set up ptr to output file name string */
     2       OPEN FILE(RECORD_FILE) OUTPUT TITLE(TEMP_STR)              /* Open the output recording file */
     2             ENVIRONMENT(MAXIMUM_RECORD_SIZE(MAX_REC_SIZE),       /*  such that others may read it */
     2                         SHARED_READ);
     2       ALLOCATE PLI_FILE_DISPLAY SET (RECORD_EXPTR);              /* Allocate space for the DISPLAY output */
     2       CALL DISPLAY (RECORD_FILE,RECORD_EXPTR->PLI_FILE_DISPLAY); /* Get the expanded file name */
     2       RECORD_STR = RECORD_EXPTR->EXPANDED_TITLE;                 /* Move the expanded string into global area for the module
     2       FREE RECORD_EXPTR->PLI_FILE_DISPLAY;                       /* Release the storage area since the expanded string has be
```

```
        2
        2          RETURN(NORMAL);
        2          END RECORD_INIT;
        1
```

```
  1     MONITOR_REQUEST: Procedure Returns(Fixed Binary(31));
  2
| 2     /*
| 2     /*      Execute first collection event. If live, collection events will
| 2     /*      continue at AST level.
| 2     /*/
  2
  2     IF ^ (MC->MCA$B_FIRSTC = PROCS_CLSNO & M->MRB$V_PLAYBACK) /* If not playback of PROCESSES */
  2         THEN DO;
  3             CALL = SYS$DCLAST(COLLECTION_EVENT,,);            /* ... then execute first collection event */
  3             IF STATUS = NOT_SUCCESSFUL                        /* $DCLAST failure? */
  3                 THEN DO;
  4                     CALL MON_ERR(MNR$_SSERROR,CALL,DCLAST_STR); /* Yes -- log the error */
  4                     RETURN(MNR$_SSERROR);                     /* ... and return with status */
  4                     END;
  3             END;
  2
| 2     /*
| 2     /*      Main monitoring loop. For playback, alternate collection and display events.
| 2     /*      For live, simply issue display events in a loop while collection events loop
| 2     /*      at AST level.
| 2     /*/
  2
  2     DO WHILE (COLLENDED = NO);                                /* Loop while collection has not ended */
  3     IF M->MRB$V_PLAYBACK                                      /* If this is a PLAYBACK request, */
  3         THEN DO;
  4             CALL = SYS$DCLAST(COLLECTION_EVENT,,);            /* ... then execute a collection event */
  4             IF STATUS = NOT_SUCCESSFUL                        /* $DCLAST failure? */
  4                 THEN DO;
  5                     CALL MON_ERR(MNR$_SSERROR,CALL,DCLAST_STR); /* Yes -- log the error */
  5                     RETURN(MNR$_SSERROR);                     /* ... and return with status */
  5                     END;
  4             IF MC->MCA$V_MULTFND & M->MRB$V_DISPLAY          /* If multiple found and display requested, */
  4                                 & COLL_STATUS = NORMAL        /* ... and collection_event finished OK, */
  4                 THEN DO;
  5                     CALL = DISPLAY_EVENT();                   /* Execute a display event */
  5                     IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Return if bad status */
  5
  5                     IF COLLENDED = NO & M->MRB$V_DISP_TO_FILE = NO /* If still collecting, and displaying to SYS$OUTPU
  5                         THEN CALL = SYS$WFLOR(0,DISP_EV_FLAG_M | REFR_EV_FLAG_M);
| 5                                                               /* ... then wait for viewing time or refresh request */
  5
  5                     END;
  4             END;
  3
  3         ELSE DO;                                              /* This is a LIVE request */
  4             IF M->MRB$V_DISPLAY                               /* If display requested */
  4                 THEN DO;
  5                     CALL = DISPLAY_EVENT();                   /* ... then execute a display request */
  5                     IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Return if bad status */
  5                     END;
  4
  4             IF COLLENDED = NO                                 /* Wait -- If no display, will wait whole request, */
  4                 THEN CALL = SYS$WFLOR(0,DISP_EV_FLAG_M | REFR_EV_FLAG_M);
  4
  4             END;                                              /* ... while collection continues at AST level */
```

```
 2389     3       END;
 2390     2
```

```
   2         /*
   2         /*      End of main monitoring loop
   2         /*/
   2
   2         RETURN(COLL_STATUS);                                   /* Return with status from COLLECTION_EVENT */
   2
   2         END MONITOR_REQUEST;
   1
```

```
     1        REQUEST_SUMMARY: Procedure Returns(Fixed Binary(31));
     2
  |  2        /*
  |  2        /*      Since the MONITOR request has terminated (except for SUMMARY),
  |  2        /*      certain CLEANUP routines may be executed now. Since SUMMARY
  |  2        /*      output uses the same SYS$OUTPUT stream through the SCRPKG as
  |  2        /*      DISPLAY output, DISPLAY_CLEANUP MUST be done now.
  |  2        /*/
     2
     2        IF M->MRB$V_RECORD & M->MRB$V_REC_CL_REQ                   /* If this is a RECORD request AND cleanup required, */
     2            THEN CALL = RECORD_CLEANUP();                          /* ... then do record cleanup */
     2        IF M->MRB$V_PLAYBACK & M->MRB$V_INP_CL_REQ                 /* If this is a PLAYBACK request AND cleanup required, */
     2            THEN CALL = INPUT_CLEANUP();                           /* ... then do cleanup for it */
     2        IF M->MRB$V_DISPLAY & M->MRB$V_DIS_CL_REQ                  /* If this is a DISPLAY request AND cleanup required, */
     2            THEN CALL = DISPLAY_CLEANUP();                         /* ... then do display cleanup */
     2
     2        CALL = SUMMARY_INIT();                                     /* Perform summary init */
     2        IF STATUS = NOT_SUCCESSFUL                                 /* Failed? */
     2            THEN DO;
     3                CALL MON_ERR(MNR$_DISPERR,CALL);                       /* Yes -- log the error */
     3                RETURN(MNR$_DISPERR);                                  /* ... and return with status */
     3                END;
     2        CALL = SUMMARY_EVENT();                                    /* Perform summarization */
     2        IF STATUS = NOT_SUCCESSFUL                                 /* If failed, then return with status */
     2            THEN RETURN(CALL);
     2
     2        RETURN(NORMAL);                                            /* Return to caller */
     2
     2        END REQUEST_SUMMARY;
     1
```

```
DISPLAY_EVENT: Procedure Returns(fixed binary(31));

/*
/*++
/*
/* FUNCTIONAL DESCRIPTION:
/*
/*      DISPLAY_EVENT
/*
/*      Called by EXECUTE_REQUEST to perform a single display event.
/*      One display event consists of creating and writing a screen
/*      image, including template if necessary, for a single class.
/*      The current class to be displayed is indicated within the
/*      DISPLAY_EVENT routine by the CURR_DCLASS variable. CURR_DCLASS
/*      is updated on each entry to DISPLAY_EVENT to indicate the
/*      next class in the list of requested classes. This causes the
/*      displays to cycle. DISPLAY_EVENT is entered once per viewing
/*      interval, or whenever a CTRL-W (screen refresh) is received.
/*
/* INPUTS:
/*
/*      None
/*
/* OUTPUTS:
/*
/*      None
/*
/* ROUTINE VALUE:
/*
/*      SS$_NORMAL, or failing MONITOR status code.
/*
/*--
/*/
```

```
 | 2      /*
 | 2      /*      |------------------------------------------------------------------------+
 | 2      /*      |                                                                        |
 | 2      /*      |                          LOCAL STORAGE                                 |
 | 2      /*      |                                                                        |
 | 2      /*      |------------------------------------------------------------------------+
 | 2      /*/
   2
   2      Declare
   2      ADV_HOM_ITEM    ENTRY (POINTER);                                   /* MACRO-32 rtn to advance homog class to next displ
   2
   2      Declare
   2      EV_FLAGS        BIT(32) ALIGNED,                                   /* Cluster 0 event flags */
   2      SS$_WASCLR      FIXED BINARY(31) GLOBALREF VALUE;                  /* "Event flag clear" return status */
   2
   2      Declare
   2      DCDB            POINTER STATIC,                                    /* CDB for current display class */
   2      COLL_TIME       BIT(64) ALIGNED STATIC,                            /* Time stamp from most recent collection */
   2      TEMP            FIXED BINARY(15);                                  /* Temporary scratch "register" */
   2
   2      Declare
   2      1 TIME_PARMS    STATIC,                                            /* FAOL parms for date and time lines */
   2        2 DATE_LEN    FIXED BINARY(31) INIT(11),                         /* Length of date string */
   2        2 DATE_PTR    POINTER,                                           /* Pointer to date string */
   2        2 TIME_LEN    FIXED BINARY(31) INIT(8),                          /* Length of time string */
   2        2 TIME_PTR    POINTER,                                           /* Pointer to time string */
   2
   2      DATE_OUT        CHAR(11) STATIC,                                   /* Date output string from ASCTIM */
   2      TIME_OUT        CHAR(8) STATIC,                                    /* Time output string from ASCTIM */
   2
   2      1 TIME_STR      GLOBALREF,                                         /* Date/time FAO control string */
   2        2 L           FIXED BINARY(7),                                   /* Length */
   2        2 S           CHAR(1),                                           /* First character of string */
   2
   2      1 SYS_TIME_STR  GLOBALREF,                                         /* SYSTEM class date/time FAO control string */
   2        2 L           FIXED BINARY(7),                                   /* Length */
   2        2 S           CHAR(1);                                           /* First character of string */
   2
   2      Declare
   2      DATA_STR        CHAR(1) BASED(DCDB->CDB$A_FAOCTR),                 /* First char of FAO ctr str for display data */
   2      FAOSTK          FIXED BINARY(31) GLOBALREF;                        /* First longword of FAOL parm list */
   2
   2      Declare
   2      VIDEO_IND       BIT(1) ALIGNED;                                    /* Video terminal indicator */
   2
```

```
    2        CALL = SYS$READEF(DISP_EV_FLAG,EV_FLAGS);                     /* Examine state of display event flag */
    2        IF STATUS = NOT_SUCCESSFUL                                    /* Failed? */
    2             THEN DO;
    3                  CALL MON_ERR(MNR$_SSERROR,CALL,READEF_STR);         /* Yes -- log the error */
    3                  RETURN(MNR$_SSERROR);                               /* ... and return with status */
    3                  END;
    2
    2        IF CALL = SS$_WASCLR                                          /* If display event flag was clear, */
    2             THEN DO;                                                 /* (Assume this is a refresh event) */
    3                  CALL = SYS$CLREF(REFR_EV_FLAG);                     /* Clear refresh event flag */
    3                  MC->MCA$V_REFRESH = YES;                            /* ... and indicate this is a refresh display event
    3                  IF STATUS = NOT_SUCCESSFUL                          /* SYS$CLREF service call failed? */
    3                       THEN DO;
    4                            CALL MON_ERR(MNR$_SSERROR,CALL,CLREF_STR);     /* Yes -- log the error */
    4                            RETURN(MNR$_SSERROR);                     /* ... and return with status */
    4                            END;
    3                  END;
    2
    2        IF CURR_DCLASS = 0 & (DISPLAYING = NO | MC->MCA$V_REFRESH = YES) /* If class data not yet displayed, AND */
  ! 2                                                                      /* ... first time thru or refresh requested */
    2             THEN DO;
    3                  VIDEO_IND = MC->MCA$V_VIDEO;                        /*...Get Video indicator */
    3                  CALL = DISP_TEMPLATE(D_CURR_CDBPTR(1),VIDEO_IND);   /* ... display a template for the first class, */
  ! 3                                                                      /* ... forcing output to screen if video terminal */
    3                  DISPLAYING = YES;                                   /* Indicate that display output has begun */
    3                  IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL);      /* Check call */
    3                  END;
    2
    2        IF CURR_DCLASS = MC->MCA$W_DCLASSCT                           /* If did final class on previous entry, */
    2             THEN TEMP = 1;                                           /* ... then start over at first one */
    2             ELSE TEMP = CURR_DCLASS + 1;                             /* ... otherwise, advance to next class */
    2        IF MC->MCA$L_COLLCNT >= 2 | D_CURR_CLASS_NO(TEMP) = PROCS_CLSNO /* If at least 2 collections have passed OR ... */
  ! 2                                                                      /* ... this is the PROCESSES class, */
    2             THEN DO;
    3                  IF ^ REPT_TOP                                       /* If not the special TOP repeat, */
    3                       THEN IF ADVANCE_DCLASS() = YES                 /* Test if display class should be advanced */
    3                                 THEN CURR_DCLASS = TEMP;             /* ... and advance it accordingly */
    3
    3                  DCDB = D_CURR_CDBPTR(CURR_DCLASS);                  /* Get CDB for current display class */
    3
    3                  IF MC->MCA$L_DISPCNT ^= 0 & (MC->MCA$V_REFRESH = YES | MC->MCA$W_DCLASSCT ^= 1) & ^ REPT_TOP
  ! 3                                                                      /* If template not printed above, AND ... */
  ! 3                                                                      /* ... refresh requested OR more than 1 class, */
  ! 3                                                                      /* ... AND not the special TOP repeat */
    3                       THEN DO;
    4                            CALL = DISP_TEMPLATE(DCDB,NO);            /* Display template */
    4                            IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL); /* Check call */
    4                            END;
    3
    3                  REPT_TOP = NO;                                      /* Eliminate future TOP repeat */
    3                  IF MC->MCA$L_DISPCNT = 0 & D_CURR_CLASS_NO(CURR_DCLASS) = PROCS_CLSNO & DCDB->CDB$B_ST ^= REG_PROC
    3                       THEN REPT_TOP = YES;                           /* If 1st TOP display, allow a 2nd consec TOP */
    3
    3                  IF CTRLCZ_HIT = NO | M->MRB$V_DISP_TO_FILE THEN     /* If CTRL-C and Z not hit OR displaying to a file,
    3                  DO;                                                 /* ... then prepare to display actual data */
    4                  IF DCDB->CDB$V_HOMOG THEN CALL ADV_HOM_ITEM(DCDB);  /* If homog class, advance to next display item */
```

```
        4                   CALL = SYS$SETAST(DISABLE_AST);                               /* Disable collection events while filling display b
        4                   CALL = FILL_DISP_BUFF(DCDB,COLL_TIME);                        /* Fill display buffer for this class */
        4                   CALL = SYS$SETAST(ENABLE_AST);                                /* Re-enable collection events */
        4
      | 4         /*
      | 4         /*        Call DISPLAY_PUT to first display the date and time of the most recent collection,
      | 4         /*        then to display the actual data itself.
      | 4         /*/
        4
```

```
     4                  CALL = SYS$ASCTIM(,DATE_OUT,COLL_TIME,0);                    /* Get ASCII date */
     4                  CALL = SYS$ASCTIM(,TIME_OUT,COLL_TIME,1);                    /* Get ASCII time */
     4                  DATE_PTR = ADDR(DATE_OUT);                                   /* Address of date string into FAOL list */
     4                  TIME_PTR = ADDR(TIME_OUT);                                   /* Address of time string into FAOL list */
     4                  FAOL_REQUESTED = YES;                                        /* Run it through FAOL */
     4                  OUTP_REQUESTED = NO;                                         /* ... but don't output it yet */
     4                  IF DCDB->CDB$V_SYSCLS & DCDB->CDB$B_ST ^= ALL_STAT           /* If special SYSTEM screen, issue it one way, */
     4                      THEN DO;
     5                           PUT_LEN = SYS_TIME_STR.L;                           /* Length of time control string */
     5                           CALL = DISPLAY_PUT(DPUT_FLAGS,PUT_LEN,SYS_TIME_STR.S,TIME_PARMS);
 :   5                                                                               /* Send date and time to SCRPKG */
     5                           END;
     4                      ELSE DO;                                                 /* Otherwise issue it another way */
     5                           PUT_LEN = TIME_STR.L;                               /* Length of time control string */
     5                           CALL = DISPLAY_PUT(DPUT_FLAGS,PUT_LEN,TIME_STR.S,TIME_PARMS);
 :   5                                                                               /* Send date and time to SCRPKG */
     5                           END;
     4                  IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL);                /* Check status */
     4
 :   4        /*
 :   4        /*      Put actual display data
 :   4        /*/
     4
     4                  IF DCDB->CDB$V_STD                                           /* Is this a standard class? */
     4                      THEN                                                     /* Standard Class */
     4                           IF DCDB->CDB$V_HOMOG                                /* Check type of standard class */
     4                               THEN DO;                                        /* Homogeneous Standard Class */
     5                                    CALL = DISPLAY_HOMOG(DCDB);                /* Send homog data display lines to SCRPKG */
     5                                    IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check status */
     5                                    END;
     4                               ELSE DO;                                        /* Heterogeneous Standard Class */
     5                                    FAOL_REQUESTED = YES;                      /* Run it through FAOL */
     5                                    OUTP_REQUESTED = YES;                      /* Output it now */
     5                                    CALL = DISPLAY_PUT(DPUT_FLAGS,DCDB->CDB$L_FAOCTR,DATA_STR,FAOSTK);
 :   5                                                                               /* Send display data to SCRPKG */
     5                                    IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL);       /* Check status */
     5                                    END;
     4                      ELSE                                                     /* Non-standard Class (PROCESSES) */
     4                           IF DCDB->CDB$B_ST = REG_PROC
     4                               THEN DO;                                        /* Regular PROCESSES display */
     5                                    CALL = DISPLAY_PROCS(DCDB,COLL_TIME);             /* Send process display lines to SCRPKG */
     5                                    IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check status */
     5                                    END;
     4                               ELSE DO;                                        /* TOP PROCESSES display */
     5                                    CALL = DISPLAY_TOP(DCDB);                  /* Send top process display lines to SCRPKG */
     5                                    IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check status */
     5                                    END;
     4                  MC->MCA$L_DISPCNT = MC->MCA$L_DISPCNT + 1;                   /* Count this display event */
     4                  END;
     3                  END;
     2
     2        IF MC->MCA$V_REFRESH THEN CALL = SYS$CANTIM(DISP_EV_FLAG,);            /* If a refresh event, cancel "regular" display time
     2
     2        IF COLLENDED = NO & ^ (M->MRB$V_PLAYBACK & M->MRB$V_DISP_TO_FILE)
 :   2                                                                               /* If collection still going, ... */
 :   2                                                                               /* ... AND not playing back to a file, */
```

```
   2              THEN DO;
   3                   CALL = SYS$SETIMR(DISP_EV_FLAG,VIEWING_DEL,,DISP_EV_FLAG); /* Set flag when ready to display again */
   3                   IF STATUS = NOT_SUCCESSFUL                             /* Failed? */
   3                       THEN DO;
   4                            CALL MON_ERR(MNR$_SSERROR,CALL,SETIMR_STR);   /* Yes -- log the error */
   4                            RETURN(MNR$_SSERROR);                         /* ... and return with status */
   4                            END;
   3                   END;
   2
   2          MC->MCA$V_REFRESH = NO;                                        /* Indicate not a refresh display event for next tim
   2
   2          RETURN(NORMAL);
   2
   2
```

```
     2      ADVANCE_DCLASS: Procedure Returns(Bit(1) aligned);              /* Test if display class should be advanced */
     3
  |  3      /*
  |  3      /*++
  |  3      /*
  |  3      /* FUNCTIONAL DESCRIPTION:
  |  3      /*
  |  3      /*      ADVANCE_DCLASS
  |  3      /*
  |  3      /*      This routine checks whether the current display class
  |  3      /*      (as indicated in the variable CURR_DCLASS) should be
  |  3      /*      advanced to the next requested class, or left where
  |  3      /*      it is. Normally, the class is advanced, but in the case
  |  3      /*      where the current class is homogeneous and not yet at
  |  3      /*      the end of its item list, the class is not advanced.
  |  3      /*
  |  3      /* INPUTS:
  |  3      /*
  |  3      /*      None
  |  3      /*
  |  3      /* OUTPUTS:
  |  3      /*
  |  3      /*      None
  |  3      /*
  |  3      /* ROUTINE VALUE:
  |  3      /*
  |  3      /*      YES if the current display class should be advanced.
  |  3      /*      NO  otherwise
  |  3      /*
  |  3      /*--
  |  3      /**/
     3
  |  3      /*
  |  3      /*      |-----------------------------------------------------------------+
  |  3      /*      |                                                                 |
  |  3      /*      |                          LOCAL STORAGE                          |
  |  3      /*      |                                                                 |
  |  3      /*      |-----------------------------------------------------------------+
  |  3      /**/
     3
     3      Declare
     3        ADVANCE_CLASS BIT(1) ALIGNED,                               /* YES => advance display class */
     3        RCDB          POINTER;                                      /* CDB pointer for most recent class */
     3
     3      ADVANCE_CLASS = YES;                                          /* Assume class will be advanced */
     3      IF CURR_DCLASS ^= 0                                           /* If not the first display event, */
     3         THEN DO;
     4              RCDB = D_CURR_CDBPTR(CURR_DCLASS);                    /*    get CDB addr for most recent display event */
     4              IF RCDB->CDB$V_HOMOG                                  /*    check if it is a homogeneous class */
     4                 THEN IF RCDB->CDB$A_CDX->CDX$B_IDISCONSEC < RCDB->CDB$A_CDX->CDX$B_IDISCT /* All items displayed? */
     4                        THEN ADVANCE_CLASS = NO;                                        /* No -- don't advance */
     4              END;
     3
     3      RETURN(ADVANCE_CLASS);                                        /* Return with indicator */
     3
     3      END ADVANCE_DCLASS;
```

```
      2
      2      END DISPLAY_EVENT;
      1
```

```
    1      SUMMARY_EVENT: Procedure Returns(fixed binary(31));
    2
  | 2      /*
  | 2      /*++
  | 2      /*
  | 2      /* FUNCTIONAL DESCRIPTION:
  | 2      /*
  | 2      /*       SUMMARY_EVENT
  | 2      /*
  | 2      /*       Called by EXECUTE_REQUEST once per request to create a
  | 2      /*       summary file containing a screen image for each of the
  | 2      /*       requested classes.
  | 2      /*
  | 2      /* INPUTS:
  | 2      /*
  | 2      /*       None
  | 2      /*
  | 2      /* OUTPUTS:
  | 2      /*
  | 2      /*       None
  | 2      /*
  | 2      /* ROUTINE VALUE:
  | 2      /*
  | 2      /*       SS$_NORMAL, or failing MONITOR status code.
  | 2      /*
  | 2      /*--
  | 2      /**/
    2
```

```
| 2     /*
| 2     /*    |---------------------------------------------------------------------------+
| 2     /*    |                                                                           |
| 2     /*    |                              LOCAL STORAGE                                |
| 2     /*    |                                                                           |
| 2     /*    |---------------------------------------------------------------------------+
| 2     /**/
  2
  2     Declare
  2     SUMMARY_TOP      ENTRY (POINTER)                                          /* MACRO-32 rtn to set up for TOP summary */
  2                      RETURNS (FIXED BINARY(31)),
  2     ADV_HOM_ITEM     ENTRY (POINTER);                                         /* MACRO-32 rtn to advance homog class to next displ
  2
  2     Declare
  2     DCDB             POINTER STATIC,                                          /* CDB for current display class */
  2     COLL_TIME        BIT(64) ALIGNED STATIC;                                  /* Time stamp from most recent collection */
  2
  2     Declare
  2     1 SUMM_PARMS     STATIC,                                                  /* FAOL parms for summary beg and end date/times */
  2       2 BEG_LEN      FIXED BINARY(31) INIT(20),                               /* Length of beginning date/time string */
  2       2 BEG_PTR      POINTER,                                                 /* Pointer to beginning date/time string */
  2       2 END_LEN      FIXED BINARY(31) INIT(20),                               /* Length of ending date/time string */
  2       2 END_PTR      POINTER,                                                 /* Pointer to ending date/time string */
  2
  2     BEG_OUT CHAR(23) STATIC,                                                  /* Beg date/time output string from ASCTIM */
  2     END_OUT CHAR(23) STATIC,                                                  /* End date/time output string from ASCTIM */
  2
  2     1 SUMMLINE_STR   GLOBALREF,                                               /* Summary date/time FAO control string */
  2       2 L            FIXED BINARY(7),                                         /* Length */
  2       2 S            CHAR(1),                                                 /* First character of string */
  2
  2     1 SYS_SUMMLINE_STR      GLOBALREF,                                        /* Summary date/time FAO control string for SYSTEM c
  2       2 L            FIXED BINARY(7),                                         /* Length */
  2       2 S            CHAR(1);                                                 /* First character of string */
  2
  2     Declare
  2     DATA_STR         CHAR(1) BASED(DCDB->CDB$A_FAOCTR),                       /* First char of FAO ctr str for display data */
  2     FAOSTK           FIXED BINARY(31) GLOBALREF;                              /* First longword of FAOL parm list */
  2
```

```
2        DO CURR_DCLASS = 1 TO MC->MCA$W_DCLASSCT                         /* Loop once for each requested class */
2            WHILE (MC->MCA$L_COLLCNT >= 2);                              /* ... but only if at least 2 collections */
3
3                DCDB = D_CURR_CDBPTR(CURR_DCLASS);                       /* Get CDB for current display class */
3                CALL = DISP_TEMPLATE(DCDB,NO);                           /* Send template to SCRPKG, but don't output yet */
3                IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL);           /* Check call */
3                IF (D_CURR_CLASS_NO(CURR_DCLASS) = PROCS_CLSNO           /* If PROCESSES Class with TOP screen, */
3                    & DCDB->CDB$B_ST ^= REG_PROC)
3                    ! (DCDB->CDB$V_SYSCLS & DCDB->CDB$B_ST ^= ALL_STAT) /* ... OR SYSTEM class with single stat, */
3
3                    THEN CALL = SUMMARY_TOP(DCDB);                       /* ... then do TOP setup */
3
3                IF DCDB->CDB$V_HOMOG                                     /* If homogeneous class, */
3                    THEN DO;
4                        DCDB->CDB$A_CDX->CDX$B_IDISCONSEC = 0;           /* Init consec display item number */
4                        DO WHILE(DCDB->CDB$A_CDX->CDX$B_IDISCONSEC < DCDB->CDB$A_CDX->CDX$B_IDISCT);
5
5                        CALL ADV_HOM_ITEM(DCDB);                         /* Advance to next display item */
5                        CALL = SUMM_ONE_CLASS();                         /* Summarize once for each item */
5                        IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL); /* Check call */
5                        END;
4                        END;
3
3                    ELSE DO;                                             /* Heterogeneous class or PROCESSES */
4                        CALL = SUMM_ONE_CLASS();                         /* Only need to call once */
4                        IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL); /* Check call */
4                        END;
3
3        END;
2
2        RETURN(NORMAL);                                                  /* Return */
2
```

```
SUMM_ONE_CLASS: Procedure Returns(fixed binary(31));

/*
/*++
/*
/* FUNCTIONAL DESCRIPTION:
/*
/*      SUMM_ONE_CLASS
/*
/*      Called by SUMMARY_EVENT to put screen images to the
/*      summary file for a single class. For heterogeneous
/*      classes, a single screen image is required. For
/*      other classes, multiple screen images may be
/*      required.
/*
/* INPUTS:
/*
/*      None
/*
/* OUTPUTS:
/*
/*      None
/*
/* ROUTINE VALUE:
/*
/*      SS$_NORMAL, or failing MONITOR status code.
/*
/*--
/*/
```

```
     3
     3              CALL = FILL_DISP_BUFF(DCDB,COLL_TIME);                      /* Fill display buffer for this class */
     3
 !   3         /*
 !   3         /*      Call DISPLAY_PUT to first display the summary time range,
 !   3         /*      then to display the actual data itself.
 !   3         /*/
     3
     3              CALL = SYS$ASCTIM(,BEG_OUT,M->MRB$Q_BEGINNING,0);           /* Get ASCII beginning time */
     3              CALL = SYS$ASCTIM(,END_OUT,COLL_TIME,0);                    /* Get ASCII ending time */
     3              BEG_PTR = ADDR(BEG_OUT);                                    /* Address of beg string into FAOL list */
     3              END_PTR = ADDR(END_OUT);                                    /* Address of end string into FAOL list */
     3              FAOL_REQUESTED = YES;                                       /* Run it through FAOL */
     3              OUTP_REQUESTED = NO;                                        /* ... but don't output it yet */
     3              IF DCDB->CDB$V_SYSCLS & DCDB->CDB$B_ST ^= ALL_STAT          /* If special SYSTEM screen, issue it a special way,
     3                  THEN DO;
     4                      PUT_LEN = SYS_SUMMLINE_STR.L;                       /* Length of SYSTEM summary control string */
     4                      CALL = DISPLAY_PUT(DPUT_FLAGS,PUT_LEN,SYS_SUMMLINE_STR.S,SUMM_PARMS);
 !   4                                                                          /* Send summary line to SCRPKG */
     4                      END;
     3                  ELSE DO;                                                /* else issue it the normal way */
     4                      PUT_LEN = SUMMLINE_STR.L;                           /* Length of summary control string */
     4                      CALL = DISPLAY_PUT(DPUT_FLAGS,PUT_LEN,SUMMLINE_STR.S,SUMM_PARMS);
 !   4                                                                          /* Send summary line to SCRPKG */
     4                      END;
     3              IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL);               /* Check status */
     3
 !   3         /*
 !   3         /*      Put actual display data
 !   3         /*/
     3
     3              IF DCDB->CDB$V_STD                                          /* Is this a standard class? */
     3                  THEN                                                    /* Standard Class */
     3                      IF DCDB->CDB$V_HOMOG                                /* Check type of standard class */
     3                          THEN DO;                                        /* Homogeneous Standard Class */
     4                              CALL = DISPLAY_HOMOG(DCDB);                 /* Send homog data display lines to SCRPKG */
     4                              IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check status */
     4                              END;
     3                          ELSE DO;                                        /* Heterogeneous Standard Class */
     4                              FAOL_REQUESTED = YES;                       /* Run it through FAOL */
     4                              OUTP_REQUESTED = YES;                       /* Output it now */
     4                              CALL = DISPLAY_PUT(DPUT_FLAGS,DCDB->CDB$L_FAOCTR,DATA_STR,FAOSTK);
 !   4                                                                          /* Send display data to SCRPKG */
     4                              IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL);         /* Check status */
     4                              END;
     3                  ELSE                                                    /* Non-standard Class (PROCESSES) */
     3                      IF DCDB->CDB$B_ST = REG_PROC
     3                          THEN DO;                                        /* Regular PROCESSES display */
     4                              CALL = DISPLAY_PROCS(DCDB,COLL_TIME);             /* Send process display lines to SCRPKG */
     4                              IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check status */
     4                              END;
     3                          ELSE DO;                                        /* TOP PROCESSES display */
     4                              CALL = DISPLAY_TOP(DCDB);                   /* Send top process display lines to SCRPKG */
     4                              IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check status */
     4                              END;
     3
```

```
      3              RETURN(NORMAL);
      3
      3              END SUMM_ONE_CLASS;
      2
      2      END SUMMARY_EVENT;
      1
```

```
     1       SAVE_SUM_BUFFS: Procedure Returns(Fixed Binary(31));                 /* Save SUM buffers into M.F. Summary Buffer */
 :   2                                                                             /* ... for all classes for the current input file */
     2
 :   2       /*
 :   2       /*++
 :   2       /*
 :   2       /* FUNCTIONAL DESCRIPTION:
 :   2       /*
 :   2       /*      SAVE_SUM_BUFFS
 :   2       /*
 :   2       /*      Called by EXECUTE_REQUEST once per request to save the
 :   2       /*      SUM buffers of all classes into their respective Multi-File
 :   2       /*      Summary Blocks.
 :   2       /*
 :   2       /* IMPLICIT INPUTS:
 :   2       /*
 :   2       /*      MFSPTR -- Pointer to MFS (Multi-File Summary Block)
 :   2       /*
 :   2       /*      MFS$B_CUR_COL -- column number for column (on m.f. summary report) into which the
 :   2       /*                       data from the SUM buffers will be stored.
 :   2       /*
 :   2       /*      DISPLAY_CLASSES -- 128-bit string of classes to be summarized (excludes STATES, MODES
 :   2       /*                         and PROCESSES if they are present only in support of SYSTEM).
 :   2       /*
 :   2       /* OUTPUTS:
 :   2       /*
 :   2       /*      None
 :   2       /*
 :   2       /* ROUTINE VALUE:
 :   2       /*
 :   2       /*      SS$_NORMAL, or failing MONITOR status code.
 :   2       /*
 :   2       /*--
 :   2       /*/
     2
 2       /*
 2       /*          |----------------------------------------------------------------------------+
 2       /*          |                                                                            |
 2       /*          |                              LOCAL STORAGE                                 |
 2       /*          |                                                                            |
 2       /*          |----------------------------------------------------------------------------+
 2       /*/
 2
 2       Declare
 2       COL_NO           FIXED BINARY(7);                                        /* Column number to store sums into */
 2
 2       Declare
 2       ALLOC_SUMBUFS    ENTRY(BIT(128) ALIGNED)                                 /* MONITOR MACRO-32 rtn to allocate m.f. summary buf
 2                        RETURNS (FIXED BINARY(31)),
 2       CAPTURE_SUMS     ENTRY (POINTER, FIXED BINARY(7))                        /* MACRO-32 routine to move SUM buffer to M.F. Summa
 2                        RETURNS (FIXED BINARY(31)),
 2       ADV_HOM_ITEM     ENTRY (POINTER);                                        /* MACRO-32 rtn to advance homog class to next displ
 2
 2       Declare
 2       DCDB             POINTER STATIC;                                         /* CDB for current class */
 2
 2       Declare
 2       DISPLAY_CLASSES BIT(128) ALIGNED GLOBALREF;                              /* Classes to be summarized */
 2
 2
```

```
   2       CALL = ALLOC_SUMBUFS(DISPLAY_CLASSES);                              /* Allocate m.f. summary buffers (if not done yet) *
   2       IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL);                      /* Check call */
   2
   2       COL_NO = MFS$B_CUR_COL;                                             /* Get number of column currently being processed */
   2
   2       DO CURR_DCLASS = 1 TO MC->MCA$W_DCLASSCT;                           /* Loop once for each summarized class */
   3
   3               DCDB = D_CURR_CDBPTR(CURR_DCLASS);                          /* Get CDB for current class */
   3
   3               IF DCDB->CDB$L_ECOUNT ^= 0                                  /* If we have some elements, */
   3                  THEN DO;
   4                       CALL = CAPTURE_SUMS(DCDB,COL_NO);                   /* Capture SUM buffer for this class and "column" */
   4                       IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL); /* Check call */
   4                       END;
   3
   3       END;
   2
   2       RETURN(NORMAL);                                                     /* Return */
   2
   2       END SAVE_SUM_BUFFS;
   1
```

```
 1      /*
 1      /*++
 1      /*
 1      /* FUNCTIONAL DESCRIPTION:
 1      /*
 1      /*          CLEANUP Routines. RECORD_CLEANUP, SUMMARY_CLEANUP
 1      /*                            INPUT_CLEANUP, and DISPLAY_CLEANUP
 1      /*
 1      /*          Called by EXECUTE_REQUFST to close files, reset terminal
 1      /*          characteristics, and release associated resources.
 1      /*          INPUT_CLEANUP can also be called by MFSUM_REQUEST to close
 1      /*          an input file and free the allocated buffer memory.
 1      /*          SUMMARY_CLEANUP can also be called by MFSUM_REQUEST to close
 1      /*          the summary file.
 1      /*
 1      /* INPUTS:
 1      /*
 1      /*          None
 1      /*
 1      /* OUTPUTS:
 1      /*
 1      /*          None
 1      /*
 1      /* ROUTINE VALUE:
 1      /*
 1      /*          SS$_NORMAL
 1      /*
 1      /*--
 1      /*/
 1
 1      REQUEST_CLEANUP: Procedure;
 2
 2      Declare
 2      FREE_MEM          ENTRY RETURNS(FIXED BINARY(31));           /* MONITOR MACRO-32 routine to issue LIB$FREE_VM's */
 2
 2
 2      CALL = FREE_MEM();                                           /* Free virtual memory acquired for this request */
 2
 2      IF M->MRB$V_RECORD & M->MRB$V_REC_CL_REQ                     /* If this is a RECORD request AND cleanup required, */
 2          THEN CALL = RECORD_CLEANUP();                            /* ... then do record cleanup */
 2      IF M->MRB$V_PLAYBACK & M->MRB$V_INP_CL_REQ                   /* If this is a PLAYBACK request AND cleanup required, */
 2          THEN CALL = INPUT_CLEANUP();                             /* ... then do cleanup for it */
 2      IF M->MRB$V_DISPLAY & M->MRB$V_DIS_CL_REQ                    /* If this is a DISPLAY request AND cleanup required, */
 2          THEN CALL = DISPLAY_CLEANUP();                           /* ... then do display cleanup */
 2      IF M->MRB$V_SUMMARY & M->MRB$V_SUM_CL_REQ                    /* If this is a SUMMARY request AND cleanup required, */
 2          THEN CALL = SUMMARY_CLEANUP();                           /* ... then do summary cleanup */
 2
 2      RETURN;
 2      END REQUEST_CLEANUP;
 1
 1
 1      RECORD_CLEANUP: Procedure Returns(fixed binary(31));
 2
 2      Declare
 2        H               POINTER;                                   /* Pointer to file header record */
 2
```

```
   2      M->MRB$V_REC_CL_REQ = NO;                                /* Indicate record cleanup is no longer required */
   2      CLOSE FILE(RECORD_FILE);                                 /* Close the record file */
   2      IF RECCT > 0                                             /* If file is non-empty, */
   2          THEN DO;                                             /* ... then want to re-write the header */
   3              OPEN FILE(RECORD_FILE) UPDATE TITLE(RECORD_STR)  /* Re-open it to re-write header while */
   3                   ENVIRONMENT(SHARED_READ);                   /*  allowing others to read the file */
   3              READ FILE(RECORD_FILE) SET(H);                   /* Read header record */
   3              H->MNR_HDR$Q_ENDING = M->MRB$Q_ENDING;           /* Update the ending time */
   3              H->MNR_HDR$L_RECCT = RECCT;                      /* ... and the record count */
   3              REWRITE FILE(RECORD_FILE);                       /* Re-write the header record */
   3              CLOSE FILE(RECORD_FILE);                         /* ... and close it up again */
   3              END;
   2
   2      RETURN(NORMAL);                                          /* Return */
   2      END RECORD_CLEANUP;
   1
   1      END EXECUTE_REQUEST;

```

```
          SUMMARY_CLEANUP: Procedure Returns(fixed binary(31));
    1
    1     %INCLUDE       MONDEF;                                /* Monitor utility structure definitions */
    1
    1     Declare
    1     LIB$SET_BUFFER  ENTRY (ANY VALUE),                    /* Rtn to set and clear buffer mode for the SCRPKG */
    1     SCR$SET_CURSOR  ENTRY (ANY VALUE, ANY VALUE),         /* SCRPKG rtn to set the cursor position */
    1     SCR$UP_SCROLL   ENTRY,                                /* SCRPKG rtn to scroll up one line */
    1     SCR$STOP_OUTPUT ENTRY;                                /* Rtn to stop SCRPKG output stream */
    1
    1     Declare
    1       NORMAL         FIXED BINARY(31) GLOBALREF,           /* MONITOR normal status value */
    1       MRBPTR         POINTER GLOBALREF,                    /* Pointer to MRB (Monitor Request Block) */
    1       M              POINTER DEFINED(MRBPTR),              /* Synonym for MRBPTR */
    1
    1     1 BOT_CURS       GLOBALREF,                            /* Place cursor on bottom of screen */
    1       2 L            FIXED BINARY(7),                      /* Length */
    1       2 S            CHAR(1),                              /* First character of string */
    1
    1     SFSPEC           CHAR(8) BASED;                        /* Dummy summary file spec descriptor */
    1
    1     M->MRB$V_SUM_CL_REQ = NO;                              /* Indicate summary cleanup is no longer required */
    1     CALL LIB$SET_BUFFER(0);                                /* Indicate "clear buffer mode" to SCRPKG */
  : 1                                                            /* ... and output what's left in the buffer */
    1     CALL SCR$SET_CURSOR(24,1);                             /* Place cursor on bottom line, */
    1     CALL SCR$UP_SCROLL();                                  /* ... and scroll up one line */
    1     CALL SCR$STOP_OUTPUT();                                /* Stop output stream and close summary file */
    1     RETURN(NORMAL);                                        /* Return */
    1     END SUMMARY_CLEANUP;
```

```
             INPUT_CLEANUP: Procedure Returns(fixed binary(31));
      1
      1      %INCLUDE         MONDEF;                                   /* Monitor utility structure definitions */
      1
      1      Declare
      1        MAX_REC_SIZE   FIXED BINARY(31) GLOBALREF VALUE,         /* Max record size for PLAYBACK & RECORD files */
      1        NORMAL         FIXED BINARY(31) GLOBALREF,               /* MONITOR normal status value */
      1        MRBPTR         POINTER GLOBALREF,                        /* Pointer to MRB (Monitor Request Block) */
      1        M              POINTER DEFINED(MRBPTR),                  /* Synonym for MRBPTR */
      1        INPUT_CPTR     POINTER GLOBALREF,                        /* Ptr to input buffer count word */
      1        INPUT_DATA     CHAR(MAX_REC_SIZE) VARYING BASED(INPUT_CPTR);   /* Playback file input buffer */
      1
      1      Declare
      1        INPUT_FILE              FILE RECORD INPUT;               /* Monitor Input (Playback) File */
      1
      1      M->MRB$V_INP_CL_REQ = NO;                                  /* Indicate input cleanup is no longer required */
      1      CLOSE FILE(INPUT_FILE);                                    /* Close the input file */
      1      IF INPUT_CPTR ^= NULL()                                    /* If input buffer had been acquired */
      1          THEN FREE INPUT_CPTR->INPUT_DATA;                      /* ... then free it */
      1      RETURN(NORMAL);                                            /* Return */
      1      END INPUT_CLEANUP;

             DISPLAY_CLEANUP: Procedure Returns(fixed binary(31));
      1
      1      %INCLUDE         MONDEF;                                   /* Monitor utility structure definitions */
      1
      1      Declare
      1      DISPLAYING       BIT(1) ALIGNED  GLOBALREF,                /* YES=> display output is active */
      1      CTRLZ_HIT        BIT(1) ALIGNED  GLOBALREF,                /* YES=> CTRL/Z has been hit */
      1      NORMAL           FIXED BINARY(31) GLOBALREF,               /* MONITOR normal return status */
      1      MRBPTR           POINTER          GLOBALREF,               /* Pointer to MRB (Monitor Request Block) */
      1      M                POINTER DEFINED(MRBPTR);                  /* Synonym for MRBPTR */
      1
      1      Declare
      1      LIB$SET_BUFFER   ENTRY (ANY VALUE),                        /* Rtn to set and clear buffer mode for the SCRPKG */
      1      PUT_TO_SCREEN    ENTRY (ANY VALUE, ANY),                   /* Rtn to put an arbitrary buffer to the SCRPKG */
      1      SCR$SET_CURSOR   ENTRY (ANY VALUE, ANY VALUE),             /* SCRPKG rtn to set the cursor position */
      1      SCR$ERASE_PAGE   ENTRY (ANY VALUE, ANY VALUE),             /* SCRPKG rtn to home the cursor & clear the entire screen *
      1      SCR$UP_SCROLL    ENTRY,                                    /* SCRPKG rtn to scroll up one line */
      1      SCR$STOP_OUTPUT  ENTRY;                                    /* Rtn to stop SCRPKG output stream */
      1
      1      Declare
      1      1 FIN_SEQ        GLOBALREF,                                /* Finish escape sequence for display terminal */
      1        2 L            FIXED BINARY(7),                          /* Length */
      1        2 S            CHAR(1),                                  /* First character of string */
      1
      1      1 BOT_CURS       GLOBALREF,                                /* Place cursor on bottom of screen */
      1        2 L            FIXED BINARY(7),                          /* Length */
      1        2 S            CHAR(1),                                  /* First character of string */
      1
      1      DFSPEC           CHAR(8) BASED;                            /* Dummy display file spec descriptor */
      1
      1      M->MRB$V_DIS_CL_REQ = NO;                                  /* Indicate display cleanup is no longer required */
      1      CALL LIB$SET_BUFFER(0);                                    /* Indicate "clear buffer mode" to SCRPKG */
    : 1                                                                 /* ... and output what's left in the buffer */
```

```
  1          CALL PUT_TO_SCREEN(FIN_SEQ.L,FIN_SEQ.S);                     /* Call SCRPKG for finish sequence, */
  1          IF DISPLAYING = YES                                          /* If actual output has begun, */
  1             THEN DO;
  2                  DISPLAYING = NO;                                     /* Indicate display output has stopped, */
  2                  CALL SCR$SET_CURSOR(24,1);                           /* ... place cursor on bottom line, */
  2                  CALL SCR$UP_SCROLL();                                /* ... and scroll up one line */
  2                  END;
  1          CALL SCR$STOP_OUTPUT();                                      /* Stop output stream if present */
  1          RETURN(NORMAL);                                              /* Return */
  1          END DISPLAY_CLEANUP;
```

```
INPUT_INIT: Procedure Returns(Fixed Binary(31));

/*
/*++
/*
/* FUNCTIONAL DESCRIPTION:
/*
/*          INPUT_INIT
/*
/*          Called by REQUEST_INIT or MFSUM_REQUEST to open the input
/*          (playback) file, performing various sanity checks on it.
/*
/* IMPLICIT INPUTS:
/*
/*          MRBPTR (or M) points to active MRB. In particular, MRB$A_INPUT
/*          points to string descriptor of file-spec to be opened.
/*
/* IMPLICIT OUTPUTS:
/*
/*          Input file has been opened.
/*          H and MCA$A_INPUT_PTR both point to first data byte of header record.
/*          MCA$L_INPUT_LEN contains length of header record.
/*
/* ROUTINE VALUE:
/*
/*          SS$_NORMAL, or failing MONITOR status code.
/*
/* SIDE EFFECTS:
/*
/*          /INPUT file (INPUT_FILE) is positioned to the file header record.
/*
/*--
/*/

/*
/*          |----------------------------------------------------------------+
/*          |                                                                |
/*          |                       INCLUDE  FILES                           |
/*          |                                                                |
/*          |----------------------------------------------------------------+
/*/

%INCLUDE        MONDEF;                                      /* Monitor utility structure definitions */

/*
/*          |----------------------------------------------------------------+
/*          |                                                                |
/*          |                    MESSAGE DEFINITIONS                         |
/*          |                                                                |
/*          |----------------------------------------------------------------+
/*/

Declare
MNR$_PREMEOF     FIXED BINARY(31) GLOBALREF VALUE,
MNR$_INVINPFIL   FIXED BINARY(31) GLOBALREF VALUE,
MNR$_UNSTLEV     FIXED BINARY(31) GLOBALREF VALUE;
```

```
      1
  |   1        /*
  |   1        /*     |------------------------------------------------------------------------+
  |   1        /*     |                                                                        |
  |   1        /*     |                     EXTERNAL STORAGE  DEFINITIONS                      |
  |   1        /*     |                                                                        |
  |   1        /*     |------------------------------------------------------------------------+
  |   1        /**/
      1
      1        Declare
      1          MON_ERR        ENTRY (ANY VALUE, ANY, ANY) OPTIONS(VARIABLE),  /* MONITOR MACRO-32 routine to log synchronous error
      1          MAX_REC_SIZE   FIXED BINARY(31) GLOBALREF VALUE,         /* Max record size for PLAYBACK & RECORD files */
      1          NORMAL         FIXED BINARY(31) GLOBALREF,               /* MONITOR normal status value */
      1          MRBPTR         POINTER GLOBALREF,                        /* Pointer to MRB (Monitor Request Block) */
      1          M              POINTER DEFINED(MRBPTR),                  /* Synonym for MRBPTR */
      1          MCAPTR         POINTER GLOBALREF,                        /* Pointer to MCA (Monitor Communication Area) */
      1          MC             POINTER DEFINED(MCAPTR),                  /* Synonym for MCAPTR */
      1          H              POINTER GLOBALREF,                        /* Pointer to input file header */
      1          ST_LEVEL_CUR   CHAR(8) GLOBALREF,                        /* Current MONITOR recording file structure level */
      1          ST_LEVEL_PB    CHAR(8) GLOBALDEF,                        /* MONITOR recording file structure level from input file */
      1          NEXT_REC       FIXED BINARY(31) GLOBALREF VALUE,         /* Read next record indicator for READ_INPUT rtn */
      1          HEADER_TYPE    FIXED BINARY(15) GLOBALREF,               /* Type for MONITOR recording file header */
      1          INPUT_CPTR     POINTER GLOBALREF,                        /* Ptr to input buffer count word */
      1          INPUT_DATA     CHAR(MAX_REC_SIZE) VARYING BASED(INPUT_CPTR);   /* Playback file input buffer */
      1
      1        Declare
      1          INPUT_FILE              FILE RECORD INPUT;               /* Monitor Input (Playback) File */
      1
  |   1        /*
  |   1        /*     |------------------------------------------------------------------------+
  |   1        /*     |                                                                        |
  |   1        /*     |                            LOCAL STORAGE                               |
  |   1        /*     |                                                                        |
  |   1        /*     |------------------------------------------------------------------------+
  |   1        /**/
      1
      1        Declare
      1
      1          TEMP_TYPE      BIT(8) ALIGNED,                                   /* Temporary area for record type byte */
      1          TEMP_PTR       POINTER,
      1          01 TEMP BASED(TEMP_PTR),
      1             02 L        FIXED BINARY(15),
      1             02 DC       FIXED BINARY(15),
      1             02 A        POINTER,
      1          TEMP_STR       CHAR(TEMP.L) BASED(TEMP.A);
      1
      1        Declare
      1          TEMP_INPUT_PTR FIXED BINARY(31) BASED(ADDR(MC->MCA$A_INPUT_PTR)); /* Alias for MCA$A_INPUT_PTR for computation */
      1
      1
      1        M->MRB$V_INP_CL_REQ = YES;                                /* Indicate input cleanup is required */
      1        CLOSE FILE(INPUT_FILE);                                   /* Make sure file is closed before opening */
      1        INPUT_CPTR = NULL();                                      /* Indicate no input buffer yet */
      1        TEMP_PTR = M->MRB$A_INPUT;                                /* Set up ptr to input file name string */
      1        OPEN FILE(INPUT_FILE) TITLE(TEMP_STR)                     /* Open the input recording file for playback */
      1            ENVIRONMENT(SHARED_WRITE);                            /*  and shared read (but have to use SHARED_WRITE) */
```

```
     1          ALLOCATE INPUT_DATA;                                            /* Allocate space for input buffer (for life of request) */
     1          MC->MCA$A_INPUT_PTR = INPUT_CPTR;                               /* Get ptr to first byte of input buffer */
     1          TEMP_INPUT_PTR = TEMP_INPUT_PTR + 2;                            /* Advance ptr beyond length word */
     1          CALL READ_INPUT(NEXT_REC);                                      /* Read first (file header) record */
     1          IF MC->MCA$V_EOF                                                /* If end-of-file, */
     1              THEN DO;
     2                  CALL MON_ERR(MNR$_PREMEOF);                             /* Can't find file header; log the error */
     2                  RETURN (MNR$_PREMEOF);                                  /* ... and return to caller */
     2                  END;
     1
     1          H = MC->MCA$A_INPUT_PTR;                                        /* Establish ptr to file header */
     1          TEMP_TYPE = UNSPEC(HEADER_TYPE);                                /* Get header type into a byte for compare */
     1          IF H->MNR_HDR$B_TYPE ^= TEMP_TYPE |                             /* If first record is not a file header or ... */
     1             SUBSTR(H->MNR_HDR$T_LEVEL,1,3) ^= SUBSTR(ST_LEVEL_CUR,1,3) /* ... MONITOR ID is not OK, */
     1              THEN DO;
     2                  CALL MON_ERR(MNR$_INVINPFIL);                           /* Log an error */
     2                  RETURN(MNR$_INVINPFIL);                                 /* ... and return to caller */
     2                  END;
     1
     1          IF SUBSTR(H->MNR_HDR$T_LEVEL,7,2) ^= SUBSTR(ST_LEVEL_CUR,7,2) /* If format level is not OK, */
     1              THEN DO;
     2                  CALL MON_ERR(MNR$_UNSTLEV);                             /* Log an error */
     2                  RETURN(MNR$_UNSTLEV);                                   /* ... and return to caller */
     2                  END;
     1
     1          ST_LEVEL_PB = H->MNR_HDR$T_LEVEL;                               /* Save playback structure level */
     1
     1          RETURN(NORMAL);
     1          END INPUT_INIT;

```

```
          DISP_TEMPLATE: Procedure (DCDB, OUTPUT_IND)
                                   Returns(Fixed Binary(31));
     1
  |  1    /*
  |  1    /*++
  |  1    /*
  |  1    /* FUNCTIONAL DESCRIPTION:
  |  1    /*
  |  1    /*      DISP_TEMPLATE
  |  1    /*
  |  1    /*      Called by DISPLAY_EVENT, SUMMARY_EVENT and PUT_SUMM_PAGE to
  |  1    /*      form and write a template for the indicated class. The template
  |  1    /*      consists of everything on the screen except actual data. This
  |  1    /*      includes the first 7 lines of the screen, the footing line and
  |  1    /*      the line item identifiers. If a bar graph has been requested,
  |  1    /*      the graph box is also included.
  |  1    /*
  |  1    /* INPUTS:
  |  1    /*
  |  1    /*      DCDB         -- Pointer to the CDB (Class Descriptor Block)
  |  1    /*                      of the class to be displayed.
  |  1    /*
  |  1    /*      OUTPUT_IND   -- Output indicator bit. If set, DISP_TEMPLATE
  |  1    /*                      sends terminal display commands to the SCRPKG
  |  1    /*                      and requests it to output to the screen. If
  |  1    /*                      OUTPUT_IND is not set, DISP_TEMPLATE sends
  |  1    /*                      terminal display commands to the SCRPKG, but
  |  1    /*                      does not request immediate screen output.
  |  1    /*
  |  1    /* OUTPUTS:
  |  1    /*
  |  1    /*      None
  |  1    /*
  |  1    /* ROUTINE VALUE:
  |  1    /*
  |  1    /*      SS$_NORMAL, or failing MONITOR status code.
  |  1    /*
  |  1    /*--
  |  1    /*/
     1
```

```
 |   1      /*
 |   1      /*      |-------------------------------------------------------------------+
 |   1      /*      |                                                                   |
 |   1      /*      |                          INCLUDE  FILES                           |
 |   1      /*      |                                                                   |
 |   1      /*      |-------------------------------------------------------------------+
 |   1      /*/
     1
     1      %INCLUDE        MONDEF;                                     /* Monitor utility structure definitions */
     1
 |   1      /*
 |   1      /*      |-------------------------------------------------------------------+
 |   1      /*      |                                                                   |
 |   1      /*      |              EXTERNAL ROUTINE DEFINITIONS                         |
 |   1      /*      |                                                                   |
 |   1      /*      |-------------------------------------------------------------------+
 |   1      /*/
     1
     1      Declare
     1      MON_ERR         ENTRY (ANY VALUE, ANY, ANY) OPTIONS(VARIABLE),  /* MONITOR MACRO-32 routine to log synchronous error
     1      TEMPLATE        ENTRY(POINTER VALUE)                        /* BLISS rtn to output template */
     1                      RETURNS(FIXED BINARY(31)),
     1      SCR$SET_CURSOR  ENTRY (ANY VALUE, ANY VALUE),               /* SCRPKG rtn to set the cursor position */
     1      DISPLAY_PUT     ENTRY(ANY, FIXED BINARY(31), ANY, ANY)      /* MACRO-32 rtn to put a display string */
     1                      OPTIONS(VARIABLE)
     1                      RETURNS(FIXED BINARY(31));
     1
 |   1      /*
 |   1      /*      |-------------------------------------------------------------------+
 |   1      /*      |                                                                   |
 |   1      /*      |                      MESSAGE DEFINITIONS                          |
 |   1      /*      |                                                                   |
 |   1      /*      |-------------------------------------------------------------------+
 |   1      /*/
     1
     1      Declare
     1      MNR$_DISPERR    FIXED BINARY(31) GLOBALREF VALUE;
     1
```

```
   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
 ! 1      /*
 ! 1      /*      !----------------------------------------------------------------------------+
 ! 1      /*      !                                                                            !
 ! 1      /*      !                    EXTERNAL STORAGE  DEFINITIONS                           !
 ! 1      /*      !                                                                            !
 ! 1      /*      !----------------------------------------------------------------------------+
 ! 1      /*/
   1
   1      Declare
   1      VTWIDTH         FIXED BINARY(31) GLOBALREF VALUE,                       /* Width of video terminal */
   1      VTHEIGHT        FIXED BINARY(31) GLOBALREF VALUE;                       /* Height of video terminal */
   1
   1      Declare
   1      CDBPTR                  POINTER GLOBALREF,                              /* Pointer to CDB (Class Descriptor Block) */
   1      C                       POINTER DEFINED(CDBPTR),                        /* Synonym for CDBPTR */
   1      MRBPTR                  POINTER GLOBALREF,                              /* Pointer to MRB (Monitor Request Block) */
   1      M                       POINTER DEFINED(MRBPTR),                        /* Synonym for MRBPTR */
   1      MCAPTR                  POINTER GLOBALREF,                              /* Pointer to MCA (Monitor Communication Area) */
   1      MC                      POINTER DEFINED(MCAPTR),                        /* Synonym for MCAPTR */
   1      SPTR                    POINTER GLOBALREF;                              /* Pointer to SYI (System Information Area) */
   1
   1      Declare
   1      NORMAL          FIXED BINARY(31) GLOBALREF;                             /* MONITOR normal return status */
   1
   1      Declare
   1      INP_COMM_STR    CHAR(MNR_HDR$K_MAXCOMLEN) GLOBALREF,                    /* User comment string from input file */
   1      INP_COMM_LEN    FIXED BINARY(15) GLOBALREF;                             /* Actual length of comment string */
   1
```

```
1        Declare
1        1 ANNCE_STR      GLOBALREF,                          /* Announcement FAO control string */
1          2 L            FIXED BINARY(7),                    /* Length */
1          2 S            CHAR(1);                            /* First character of string */
1
1        Declare
1        1 STATUS_STR     GLOBALREF,                          /* Status FAO control string */
1          2 L            FIXED BINARY(7),                    /* Length */
1          2 S            CHAR(1),                            /* First character of string */
1        STATUS_PARMS     CHAR(12) GLOBALREF;                 /* 3 longword FAOL parms for status display */
1
1        Declare
1        1 TABHEAD_STR    GLOBALREF,                          /* Tabular heading control string */
1          2 L            FIXED BINARY(7),                    /* Length */
1          2 S            CHAR(1),                            /* First character of string */
1        TABHEAD_PARM     POINTER STATIC,                     /* FAOL parm indicating % or blank */
1        PCENT_STR        CHAR(2) GLOBALREF,                  /* Percent symbol cstring */
1        BLANK_STR        CHAR(2) GLOBALREF;                  /* Blank character cstring */
1
1        Declare
1        1 PROCHEAD_STR   GLOBALREF,                          /* PROCESSES heading control string */
1          2 L            FIXED BINARY(7),                    /* Length */
1          2 S            CHAR(1);                            /* First character of string */
1
1        Declare
1        1 MF_STATHEAD_STR GLOBALREF,                         /* M.F. summary statistic heading control string */
1          2 [            FIXED BINARY(7),                    /* Length */
1          2 S            CHAR(1);                            /* First character of string */
1
1
```

```
 1        /*
 1        /*       |-------------------------------------------------------------------------+
 1        /*       |                                                                         |
 1        /*       |                              OWN STORAGE                                |
 1        /*       |                                                                         |
 1        /*       |-------------------------------------------------------------------------+
 1        /*/
 1
 1        Declare
 1        CALL                FIXED BINARY(31) STATIC,                    /* Holds function value (return status) of called ro
 1        STATUS              BIT(1)  BASED(ADDR(CALL));                  /* Low-order status bit for called routines */
 1
 1        Declare
 1        1 DPUT_FLAGS,                                                    /* DISPLAY_PUT routine flags */
 1          2 FAOL_REQUESTED BIT(8) ALIGNED,                               /* YES => Xlate buffer with FAOL first */
 1          2 OUTP_REQUESTED BIT(8) ALIGNED,                               /* YES => Really output buffer */
 1        PUT_LEN             FIXED BINARY(31);                            /* Length of buffer for DISPLAY_PUT to put */
 1
 1        Declare
 1        DCDB                POINTER,                                     /* Pointer to current display class CDB */
 1        OUTPUT_IND          BIT(1) ALIGNED,                              /* YES => output the template */
 1        I                   FIXED BINARY(15);                            /* Index for DO loop */
 1
 1        Declare
 1        SPEC_SYSTEM_SCREEN BIT(1) ALIGNED;                               /* YES => special screen for SYSTEM class */
 1
 1        Declare
 1        1 TITLE_PARMS     STATIC,                                        /* FAOL parms for title display line */
 1          2 BLANKS        FIXED BINARY(31),                              /* Number of preceding blanks */
 1          2 TITLE_PTR     POINTER,                                       /* Pointer to title cstring */
 1          2 PCENT_WID     FIXED BINARY(31),                              /* Width of percent string (0 or 4) */
 1          2 NODE_PTR      POINTER,                                       /* Pointer to DECnet node name cstring */
 1        TITLE_LEN         FIXED BINARY(7) BASED(TITLE_PTR),              /* Length of title string */
 1        NODE_LEN          FIXED BINARY(7) BASED(NODE_PTR),               /* Length byte of node name cstring */
 1        1 TITLE_STR       GLOBALREF,                                     /* Title FAO control string */
 1          2 L             FIXED BINARY(7),                               /* Length */
 1          2 S             CHAR(1);                                       /* First character of string */
 1
 1        Declare
 1        1 COMM_PARMS      STATIC,                                        /* FAOL parms for comment display line */
 1          2 BLANKS        FIXED BINARY(31),                              /* Number of preceding blanks */
 1          2 COMM_LEN      FIXED BINARY(31),                              /* Length of comment */
 1          2 COMM_ADDR     POINTER,                                       /* Address of comment string */
 1        1 COMM_STR        GLOBALREF,                                     /* Comment FAO control string */
 1          2 L             FIXED BINARY(7),                               /* Length */
 1          2 S             CHAR(1);                                       /* First character of string */
 1
 1        Declare
 1        1 SYS_HEAD_PARMS STATIC,                                         /* FAOL parms for SYSTEM heading line */
 1          2 SYS_NODE_PTR  POINTER,                                       /* Pointer to DECnet node name cstring */
 1          2 STATLONG_LEN  FIXED BINARY(31) INIT(7),                      /* Length of requested stat */
 1          2 STATLONG_ADDR POINTER,                                       /* Addr of requested stat */
 1
 1        1 SYS_HEAD_STR    GLOBALREF,                                     /* SYSTEM class heading control string */
 1          2 L             FIXED BINARY(7),                               /* Length */
 1          2 S             CHAR(1),                                       /* First character of string */
```

```
      1
      1      STAT_LONG (4)             CHAR(7) GLOBALREF,                     /* Table of 7-char statistic strings */
      1      SYS_BOX_STR_ADDR          POINTER GLOBALREF,                     /* Address of screen box string */
      1      SYS_BOX_STR               CHAR(1) BASED(SYS_BOX_STR_ADDR),       /* SYSTEM screen boxes (1st char) */
      1      SYS_BOX_STR_LEN           FIXED BINARY(15) GLOBALREF,            /* Its length */
      1      SYS_TEXT_STR              CHAR(1) GLOBALREF,                     /* SYSTEM screen text */
      1      SYS_TEXT_STR_LEN          FIXED BINARY(15) GLOBALREF,            /* Its length */
      1      SYS_FAO_STR               CHAR(1) GLOBALREF,                     /* SYSTEM FAO string */
      1      SYS_FAO_STR_LEN           FIXED BINARY(15) GLOBALREF;            /* Its length */
      1
      1      Declare
      1      1 SYS_BOX_PARMS,                                                 /* FAOL parms for SYSTEM screen boxes */
      1         2 SBP1          FIXED BINARY(31),                             /* Free List bar range value */
      1         2 SBP2          POINTER,                                      /* Pointer to "K" or null cstring */
      1         2 SBP3          FIXED BINARY(31),                             /* Mod List bar range value */
      1         2 SBP4          POINTER;                                      /* Pointer to "K" or null cstring */
      1
      1      Declare
      1      1 BU_SYS_SINGLE           GLOBALREF,                             /* Bar graph range values for SYSTEM class (single s
      1         2 BSS_RANGE (1:17)     FIXED BINARY(31),
      1      K_STR                     CHAR(2) GLOBALREF,                     /* K symbol for box */
      1      NULL_STR                  FIXED BINARY(15) INIT(0);              /* Dummy null symbol for box */
      1
      1
```

```
     1          MC->MCA$V_ERA_SCRL = NO;                            /* Indicate no need to erase scrolling region ... */
 :   1                                                              /* ... (display area) for PROCESSES and homogs */
     1          IF DCDB->CDB$V_HOMOG
     1               THEN DCDB->CDB$A_CDX->CDX$L_PREV_DCT = 0; /* Init count of previous display elements ... */
 :   1                                                              /* ... for homogeneous class */
     1
     1          IF DCDB->CDB$V_SYSCLS & DCDB->CDB$B_ST ^= ALL_STAT /* If special SYSTEM screen, */
     1               THEN SPEC_SYSTEM_SCREEN = YES;                 /* Set a bit for quick reference */
     1               ELSE SPEC_SYSTEM_SCREEN = NO;                  /* Otherwise, turn it off */
     1
     1
 :   1          /*
 :   1          /*        Send announcement string to SCRPKG via DISPLAY_PUT routine
 :   1          /*        This string is independent of screen style.
 :   1          /*/
     1
     1          PUT_LEN = ANNCE_STR.L;                              /* Get length of this put */
     1          FAOL_REQUESTED = NO;                                /* No need to go thru $FAOL */
     1          OUTP_REQUESTED = NO;                                /* Not ready to actually output it yet */
     1          CALL = DISPLAY_PUT(DPUT_FLAGS,PUT_LEN,ANNCE_STR.S,);
 :   1                                                              /* Send announcement string to SCRPKG */
     1          IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL);       /* Check status */
     1
 :   1          /*
 :   1          /*        Send status (footing) string to SCRPKG via DISPLAY_PUT
 :   1          /*        routine. This string is independent of screen style.
 :   1          /*        Skip it, however, for multi-file summary.
 :   1          /*/
     1
     1          IF M->MRB$V_MFSUM = NO                              /* If not multi-file summary, */
     1               THEN DO;
     2                    PUT_LEN = STATUS_STR.L;                   /* Get length of this put */
     2                    FAOL_REQUESTED = YES;                     /* Request a run thru $FAOL */
     2                    OUTP_REQUESTED = NO;                      /* Not ready to actually output it yet */
     2                    CALL = DISPLAY_PUT(DPUT_FLAGS,PUT_LEN,STATUS_STR.S,STATUS_PARMS);
 :   2                                                              /* Send status string to SCRPKG */
     2                    IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL); /* Check status */
     2                    END;
     1
     1
 !   1        /*
 !   1        /*      Send title string to SCRPKG via DISPLAY_PUT routine.
 !   1        /*      Includes DECnet node name if one is present
 !   1        /*      This string is independent of screen style.
 !   1        /*/
     1
     1        TITLE_PTR = DCDB->CDB$A_TITLE;                       /* Establish title pointer */
     1        TITLE_PARMS.BLANKS = DIVIDE((VTWIDTH - TITLE_LEN),2,31) - 1; /* Compute preceding blanks */
     1        IF DCDB->CDB$V_PERCENT = YES & M->MRB$V_MFSUM = NO /* If percent requested for other than m.f. summary, */
     1            THEN PCENT_WID = 4;                              /* then put out % string */
     1            ELSE PCENT_WID = 0;                              /* else don't put % string */
     1        PUT_LEN = TITLE_STR.L;                               /* Get length of this put */
     1        NODE_PTR = ADDR(SPTR->MNR_SYI$T_NODENAME);           /* Set up ptr to node name cstring */
     1        IF NODE_LEN = 0 | SPEC_SYSTEM_SCREEN                 /* If node name non-existent, or special SYSTEM screen, */
     1            THEN PUT_LEN = PUT_LEN - 16;                     /*    then chop off node name line */
     1        FAOL_REQUESTED = YES;                                /* Request a run thru $FAOL */
     1        OUTP_REQUESTED = NO;                                 /* Not ready to actually output it yet */
     1        CALL = DISPLAY_PUT(DPUT_FLAGS,PUT_LEN,TITLE_STR.S,TITLE_PARMS);
 !   1                                                             /* Send title line to SCRPKG */
     1        IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL);   /* Check status */
     1
 !   1        /*
 !   1        /*      If special screen display for SYSTEM class,
 !   1        /*      send a heading string including DECnet node
 !   1        /*      name and requested statistic.
 !   1        /*/
     1
     1        IF SPEC_SYSTEM_SCREEN
     1           THEN DO;
     2                STATLONG_ADDR = ADDR(STAT_LONG(DCDB->CDB$B_ST));            /* Get addr of correct stat string */
     2                SYS_NODE_PTR = NODE_PTR;                                     /* Get address of node name cstring */
     2                PUT_LEN = SYS_HEAD_STR.L;                                    /* Get length of this put */
     2                FAOL_REQUESTED = YES;                                        /* Request a run thru $FAOL */
     2                OUTP_REQUESTED = NO;                                         /* Not ready to actually output it yet */
     2                CALL = DISPLAY_PUT(DPUT_FLAGS,PUT_LEN,SYS_HEAD_STR.S,SYS_HEAD_PARMS);
 !   2                                                                             /* Send SYSTEM heading to SCRPKG */
     2                IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL);                /* Check status */
     2                END;
     1
 !   1        /*
 !   1        /*      Send user's comment string to SCRPKG via DISPLAY_PUT routine.
 !   1        /*      This string is independent of screen style.
 !   1        /*/
     1
     1        IF M->MRB$V_MFSUM = NO & INP_COMM_LEN ^= 0              /* If not m.f. summary and an input comment exists, */
     1           & SPEC_SYSTEM_SCREEN = NO                             /* and not the special SYSTEM screen, */
     1          THEN DO;
     2                COMM_LEN = INP_COMM_LEN;                         /* Move length to parm list */
     2                COMM_ADDR = ADDR(INP_COMM_STR);                  /* Move address to parm list */
     2                COMM_PARMS.BLANKS = DIVIDE((VTWIDTH - COMM_LEN),2,31) - 1; /* Compute preceding blanks */
     2                PUT_LEN = COMM_STR.L;                            /* Get length of this put */
     2                FAOL_REQUESTED = YES;                            /* Request a run thru $FAOL */
     2                OUTP_REQUESTED = NO;                             /* Not ready to actually output it yet */
     2                CALL = DISPLAY_PUT(DPUT_FLAGS,PUT_LEN,COMM_STR.S,COMM_PARMS);
 !   2                                                                 /* Send comment line to SCRPKG */
```

```
      2              IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL);  /* Check status */
      2              END;
      1
```

```
 ! 1          /*
 ! 1          /*      For standard classes, call TEMPLATE to put item
 ! 1          /*      names and build FAO string for actual data for
 ! 1          /*      tabular or bar-style screen. Skip, however, for
 ! 1          /*      the special SYSTEM display
 ! 1          /*/
   1
   1            IF DCDB->CDB$V_STD & SPEC_SYSTEM_SCREEN = NO              /* If standard class, and not SYSTEM screen, */
   1              THEN DO;
   2                   CALL = TEMPLATE(DCDB);                             /* Put item names and build FAO string */
   2                   IF STATUS = NOT_SUCCESSFUL                         /* Check status */
   2                       THEN DO;
   3                            CALL MON_ERR(MNR$_DISPERR,CALL);          /* Log the error */
   3                            RETURN(MNR$_DISPERR);                     /* ... and return with status */
   3                            END;
   2                   END;
   1
 ! 1          /*
 ! 1          /*      Send heading string (and box, if bar graph)
 ! 1          /*      to SCRPKG via DISPLAY_PUT routine.
 ! 1          /*/
   1
   1          IF M->MRB$V_MFSUM = NO & SPEC_SYSTEM_SCREEN = NO                     /* Only do it if not multi-file summary and not spec
   1          THEN
   1
 ! 1          /*
 ! 1          /*      Put PROCESSES Heading
 ! 1          /*/
   1
   1          IF ^ DCDB->CDB$V_STD & DCDB->CDB$B_ST = REG_PROC                    /* Put out regular PROCESSES heading */
   1              THEN DO;
   2                   PUT_LEN = PROCHEAD_STR.L;                                    /* Length of put */
   2                   FAOL_REQUESTED = NO;                                         /* No $FAOL required */
   2                   OUTP_REQUESTED = OUTPUT_IND;                                 /* Output it if caller requested */
   2                   CALL = DISPLAY_PUT(DPUT_FLAGS,PUT_LEN,PROCHEAD_STR.S,); /* Hand heading over to SCRPKG */
   2                   IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL);                /* Check status */
   2                   END;
   1
 ! 1          /*
 ! 1          /*      Put Tabular Heading
 ! 1          /*/
   1
   1              ELSE IF DCDB->CDB$V_STD & DCDB->CDB$B_ST = ALL_STAT                      /* All statistics requested for STD class? *
   1                       THEN DO;                                                       /* Tabular display */
   2                            IF DCDB->CDB$V_WIDE                                       /* If a wide display (for DISK), */
   2                                 THEN CALL SCR$SET_CURSOR(6,44);                      /* ... then set appropriate cursor */
   2                                 ELSE CALL SCR$SET_CURSOR(6,40);                      /* ... else set it to the usual place */
   2                            IF DCDB->CDB$V_PERCENT
   2                                 THEN TABHEAD_PARM = ADDR(PCENT_STR);                 /* Include % symbol in heading */
   2                                 ELSE TABHEAD_PARM = ADDR(BLANK_STR);                 /* Exclude % symbol from heading */
   2                            PUT_LEN = TABHEAD_STR.L;                                  /* Length of put */
   2                            FAOL_REQUESTED = YES;                                     /* Request a run thru $FAOL */
   2                            OUTP_REQUESTED = OUTPUT_IND;                              /* Output it if caller requested */
   2                            CALL = DISPLAY_PUT(DPUT_FLAGS,PUT_LEN,TABHEAD_STR.S,TABHEAD_PARM);
 ! 2                                                                                      /* Hand heading over to SCRPKG */
   2                            IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL);             /* Check status */
```

```
     2                      END;
     1
```

```
 ! 1        /*
 ! 1        /*      Put Bar Graph Heading
 ! 1        /*/
   1
   1             ELSE BEGIN;                                                             /* Bar graph display */
   2                  Declare
   2                  CURGR_VAL       FIXED BINARY(31),                                 /* Current graph value (for heading) */
   2                  MAXGR_VAL       FIXED BINARY(31),                                 /* Max (right-edge) graph value for heading */
   2                  GR_INCR         FIXED BINARY(31),                                 /* Increment value for heading */
   2                  RANGE           FIXED BINARY(31),                                 /* Range for heading values */
   2                  CHAR_ADDR       POINTER;                                          /* Addr of symbol char for heading */
   2
   2                  Declare
   2                  1 BARHEAD_STR   GLOBALREF,                                        /* Bar graph heading control string */
   2                    2 L           FIXED BINARY(7),                                  /* Length */
   2                    2 S           CHAR(1),                                          /* First character of string */
   2                  1 BARHEAD_PARMS,                                                  /* FAOL parms for graph heading line */
   2                    2 BP1         FIXED BINARY(31),                                 /* Graph heading value */
   2                    2 BP2         POINTER,                                          /* Graph heading symbol string ptr */
   2                    2 BP3         FIXED BINARY(31),                                 /* Graph heading value */
   2                    2 BP4         POINTER,                                          /* Graph heading symbol string ptr */
   2                    2 BP5         FIXED BINARY(31),                                 /* Graph heading value */
   2                    2 BP6         POINTER,                                          /* Graph heading symbol string ptr */
   2                    2 BP7         FIXED BINARY(31),                                 /* Graph heading value */
   2                    2 BP8         POINTER,                                          /* Graph heading symbol string ptr */
   2                    2 BP9         FIXED BINARY(31),                                 /* 1=> advance heading one byte to right */
   2                    2 BP10        FIXED BINARY(31),                                 /* Graph heading value */
   2                    2 BP11        POINTER;                                          /* Graph heading symbol string ptr */
   2
   2                  Declare
   2                  1 STATHEAD_STR GLOBALREF,                                         /* Bar graph statistic heading control string */
   2                    2 L           FIXED BINARY(7),                                  /* Length */
   2                    2 S           CHAR(1),                                          /* First character of string */
   2                  1 STATHEAD_PARMS,                                                 /* FAOL parms for statistic heading */
   2                    2 L           FIXED BINARY(31) INIT(3),                         /* Statistic heading string length */
   2                    2 A           POINTER,                                          /* Pointer to heading string */
   2                  STAT_HEAD (4)   CHAR(3) GLOBALREF;                                /* Table of 3-char heading strings */
   2
```

```
        2                CALL = PUT_BOX();                                         /* Put larger bar graph box to SCRPKG */
        2                IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL);             /* Check status */
        2
  !     2        /*
  !     2        /*     Put heading line on top of the box.
  !     2        /*/
        2
        2                IF DCDB->CDB$V_PERCENT
        2                    THEN DO;                                              /* Heading values are percents */
        3                         CHAR_ADDR = ADDR(PCENT_STR);                     /* Use % symbol for heading */
        3                         CURGR_VAL = 0;                                   /* First value is 0 */
        3                         RANGE = 100;                                     /* Range is 100 */
        3                         END;
        2                    ELSE IF DCDB->CDB$V_KUNITS
        2                            THEN DO;                                      /* Values in units of 1000 */
        3                                 CHAR_ADDR = ADDR(K_STR);                 /* Use K symbol for heading */
        3                                 CURGR_VAL = DIVIDE(DCDB->CDB$L_MIN,1000,31); /* Compute first value */
        3                                 RANGE = DIVIDE(DCDB->CDB$L_RANGE,1000,31);   /* ... and range */
        3                                 END;
        2                            ELSE DO;                                      /* Heading values are as is */
        3                                 CHAR_ADDR = ADDR(NULL_STR);              /* Use no (null) symbol for heading */
        3                                 CURGR_VAL = DCDB->CDB$L_MIN;             /* Compute first value */
        3                                 RANGE = DCDB->CDB$L_RANGE;               /* ... and range */
        3                                 END;
        2                GR_INCR = DIVIDE(RANGE,4,31);                             /* Compute increment between values */
        2                MAXGR_VAL = CURGR_VAL + RANGE;                            /* ... and max (right-most) value */
        2                BP1 = CURGR_VAL;                                          /* Fill in FAOL parms to put heading */
        2                BP2 = CHAR_ADDR;                                          /* ......... */
        2                CURGR_VAL = CURGR_VAL + GR_INCR;                          /* Compute next value */
        2                BP3 = CURGR_VAL;                                          /* ......... */
        2                BP4 = CHAR_ADDR;                                          /* ......... */
        2                CURGR_VAL = CURGR_VAL + GR_INCR;                          /* Compute next value */
        2                BP5 = CURGR_VAL;                                          /* ......... */
        2                BP6 = CHAR_ADDR;                                          /* ......... */
        2                CURGR_VAL = CURGR_VAL + GR_INCR;                          /* Compute next value */
        2                BP7 = CURGR_VAL;                                          /* ......... */
        2                BP8 = CHAR_ADDR;                                          /* ......... */
        2                CURGR_VAL = CURGR_VAL + GR_INCR;                          /* Compute next value */
        2                IF DCDB->CDB$V_PERCENT | DCDB->CDB$V_KUNITS               /* If units symbol is printable, */
        2                    THEN BP9 = 0;                                         /* ... then do not advance one space */
        2                    ELSE BP9 = 1;                                         /* ... else advance a space, so last value */
  !     2                                                                          /* ... is on right edge of box */
        2                BP10 = MAXGR_VAL;                                         /* Next parm is the last value */
        2                BP11 = CHAR_ADDR;                                         /* ... addr of units symbol */
        2
```

```
 ! 2      /*
 ! 2      /*      Setup to call DISPLAY_PUT for the heading line
 ! 2      /*/
   2               PUT_LEN = BARHEAD_STR.L;                                          /* Length of put */
   2               FAOL_REQUESTED = YES;                                             /* Request a run thru $FAOL */
   2               OUTP_REQUESTED = NO;                                              /* Not ready to output it yet */
   2               CALL = DISPLAY_PUT(DPUT_FLAGS,PUT_LEN,BARHEAD_STR.S,BARHEAD_PARMS);
 ! 2                                                                                 /* Hand heading over to SCRPKG */
   2               IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL);                     /* Check status */
 ! 2      /*
 ! 2      /*      Now put the smaller box with the statistic heading for Standard classes
 ! 2      /*/
   2               IF DCDB->CDB$V_STD                                                /* If standard class, */
   2                   THEN DO;
   3                       STATHEAD_PARMS.A = ADDR(STAT_HEAD(DCDB->CDB$B_ST));    /* Get addr of correct string */
   3                       PUT_LEN = STATHEAD_STR.L;                              /* Length of put */
   3                       FAOL_REQUESTED = YES;                                  /* Request a run thru $FAOL */
   3                       OUTP_REQUESTED = OUTPUT_IND;                           /* Output it if caller requested */
   3                       CALL = DISPLAY_PUT(DPUT_FLAGS,PUT_LEN,STATHEAD_STR.S,STATHEAD_PARMS);
 ! 3                                                                              /* Hand statistic heading over to SCRPKG */
   3                       IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL);          /* Check status */
   3                       END;
   2
   2               END;                                                              /* End of begin-end group */
   1
```

```
     1          ELSE                                                               /* At this point, either m.f.summm or spec. SYSTEM sc
     1                  IF SPEC_SYSTEM_SCREEN                                      /* If special SYSTEM screen, */
     1                  THEN
     1                  DO;
     2                  DCDB->CDB$A_FAOCTR = ADDR(SYS_FAO_STR);                    /* Get a pre-built FAO control string */
     2                  DCDB->CDB$L_FAOCTR = SYS_FAO_STR_LEN;                      /* ... and its length */
     2                  IF BSS_RANGE(14) >= 10000                                  /* If range of Free List bar is large, */
     2                    THEN DO;
     3                          SBP1 = DIVIDE(BSS_RANGE(14),1000,31);              /*   then get the number of thousands */
     3                          SBP2 = ADDR(K_STR);                                /*   and use a "K" */
     3                          END;
     2                    ELSE DO;
     3                          SBP1 = BSS_RANGE(14);                              /*   else use the raw number */
     3                          SBP2 = ADDR(NULL_STR);                             /*   and no "K" */
     3                          END;
     2
     2                  IF BSS_RANGE(15) >= 10000                                  /* If range of Modified List bar is large, */
     2                    THEN DO;
     3                          SBP3 = DIVIDE(BSS_RANGE(15),1000,31);              /*   then get the number of thousands */
     3                          SBP4 = ADDR(K_STR);                                /*   and use a "K" */
     3                          END;
     2                    ELSE DO;
     3                          SBP3 = BSS_RANGE(15);                              /*   else use the raw number */
     3                          SBP4 = ADDR(NULL_STR);                             /*   and no "K" */
     3                          END;
     2
     2                  PUT_LEN = SYS_BOX_STR_LEN;                                 /* Length of put */
     2                  FAOL_REQUESTED = YES;                                      /* Request a run thru $FAOL */
     2                  OUTP_REQUESTED = NO;                                       /* Don't output yet */
     2                  CALL = DISPLAY_PUT(DPUT_FLAGS,PUT_LEN,SYS_BOX_STR,SYS_BOX_PARMS); /* Hand boxes over to SCRPKG */
     2                  IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL);              /* Check status */
     2
     2                  PUT_LEN = SYS_TEXT_STR_LEN;                                /* Length of put */
     2                  FAOL_REQUESTED = YES;                                      /* Request a run thru $FAOL */
     2                  OUTP_REQUESTED = OUTPUT_IND;                               /* Output it if caller requested */
     2                  CALL = DISPLAY_PUT(DPUT_FLAGS,PUT_LEN,SYS_TEXT_STR,);     /* Output text and display entire screen */
     2                  IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL);              /* Check status */
     2                  END;
     1
     1                  ELSE                                                       /* Multi-file summary */
     1                  DO;
     2                  PUT_LEN = MF_STATHEAD_STR.L;                               /* Length of put */
     2                  FAOL_REQUESTED = YES;                                      /* Request a run thru $FAOL */
     2                  OUTP_REQUESTED = OUTPUT_IND;                               /* Output it if caller requested */
     2                  CALL = DISPLAY_PUT(DPUT_FLAGS,PUT_LEN,MF_STATHEAD_STR.S,); /* Hand statistic heading over to SCRPKG */
     2                  IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL);              /* Check status */
     2                  END;
     1
     1          RETURN(NORMAL);                                                    /* Return to caller */
     1
     1   PUT_BOX: Procedure Returns(fixed binary(31));
     2
   | 2   /*
   | 2   /*++
   | 2   /*
   | 2   /* FUNCTIONAL DESCRIPTION:
   | 2   /*
   | 2   /*      PUT_BOX
   | 2   /*
   | 2   /*      Called by DISP_TEMPLATE to put the bar graph box to the SCRPKG.
   | 2   /*      Actual display output is not performed.
   | 2   /*
   | 2   /* INPUTS:
   | 2   /*
   | 2   /*      None
   | 2   /*
   | 2   /* OUTPUTS:
   | 2   /*
   | 2   /*      None
   | 2   /*
   | 2   /* ROUTINE VALUE:
   | 2   /*
   | 2   /*      SS$_NORMAL, or failing MONITOR status code.
   | 2   /*
   | 2   /*--
   | 2   /*/
     2
```

```
  2      /*
  2      /*     |-----------------------------------------------------------------------+
  2      /*     |                                                                       |
  2      /*     |                         LOCAL STORAGE                                 |
  2      /*     |                                                                       |
  2      /*     |-----------------------------------------------------------------------+
  2      /*/
  2
  2      Declare
  2      FIRST_DATA_LINE          FIXED BINARY(31) GLOBALREF VALUE,    /* Line number of first data line on screen */
  2      LAST_DATA_LINE           FIXED BINARY(31) GLOBALREF VALUE,    /* Line number of last data line on screen */
  2      VTDATALINES              FIXED BINARY(31) GLOBALREF VALUE,    /* Total data lines on screen */
  2      CURSOR_STR               CHAR(2) GLOBALREF,                   /* Cursor control escape sequence */
  2      HORIZ_STR                CHAR(42) GLOBALREF,                  /* Horizontal portion of bar graph box */
  2      CURROW                   FIXED BINARY(15),                    /* Current row counter */
  2      CURCOL                   FIXED BINARY(15);                    /* Current column counter */
  2
  2      Declare
  2      1 VERT_LINE (5*VTDATALINES),                                  /* Escape string to make vertical lines for box */
  2        2 CURSOR               CHAR(2),                             /* Cursor control esc sequence */
  2        2 ROW                  FIXED BINARY(7),                     /* Row byte */
  2        2 COL                  FIXED BINARY(7),                     /* Column byte */
  2        2 VERT_CHAR            CHAR(1);                             /* Vertical bar character */
  2
  2      Declare
  2      1 HORIZ_LINE (2),                                             /* Escape string to make horiz'l lines for box */
  2        2 CURSOR               CHAR(2),                             /* Cursor control esc sequence */
  2        2 ROW                  FIXED BINARY(7),                     /* Row byte */
  2        2 COL                  FIXED BINARY(7),                     /* Column byte */
  2        2 TOP_BOT              CHAR(42);                            /* Horiz line appearing at top & bot of box */
  2
```

```
    ¦   2       /*
    ¦   2       /*      Create and send to the SCRPKG the horizontal (top
    ¦   2       /*      and bottom) lines of the bar graph box.
    ¦   2       /*/
        2
        2       HORIZ_LINE.CURSOR(1) = CURSOR_STR;              /* Move in cursor control sequence */
        2       HORIZ_LINE.ROW(1) = FIRST_DATA_LINE - 1;        /* Move in row number of top line of box */
        2       HORIZ_LINE.COL(1) = 38;                         /* Move in column number */
        2       TOP_BOT(1) = HORIZ_STR;                         /* Move in the top line */
        2       HORIZ_LINE.CURSOR(2) = CURSOR_STR;              /* Move in cursor control sequence */
        2       HORIZ_LINE.ROW(2) = LAST_DATA_LINE+1;           /* Move in row number of bot line of box */
        2       HORIZ_LINE.COL(2) = 38;                         /* Move in column number */
        2       TOP_BOT(2) = HORIZ_STR;                         /* Move in the bottom line */
        2       FAOL_REQUESTED = NO;                            /* FAOL not involved */
        2       OUTP_REQUESTED = NO;                            /* ... don't output it yet */
        2       CALL = DISPLAY_PUT(DPUT_FLAGS,46*2,HORIZ_LINE,); /* Put horizontal lines to SCRPKG */
        2       IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL);   /* Check status */
        2
    ¦   2       /*
    ¦   2       /*      Now create and send to the SCRPKG the vertical
    ¦   2       /*      lines of the bar graph box.
    ¦   2       /*/
        2
        2       I = 0;                                          /* Initialize loop control */
        2       DO CURROW = FIRST_DATA_LINE TO LAST_DATA_LINE;  /* Loop once for each data line in graph */
        3       DO CURCOL = 38 TO 78 BY 10;                     /* Loop once for each vert char in a line */
        4       IF CURCOL = 78 THEN CURCOL = CURCOL + 1;        /* Push right-most bar over 1 */
        4       I = I + 1;                                      /* Update index into VERT_LINE vector */
        4       VERT_LINE.CURSOR(I) = CURSOR_STR;               /* Move in cursor control sequence */
        4       VERT_LINE.ROW(I) = CURROW;                      /* Move in row number */
        4       VERT_LINE.COL(I) = CURCOL;                      /* Move in column number */
        4       VERT_CHAR(I) = '¦';                             /* Move in the vertical bar char */
        4       END;
        3       END;
        2       CALL = DISPLAY_PUT(DPUT_FLAGS,5*5*VTDATALINES,VERT_LINE,); /* Put vertical lines to SCRPKG */
        2       IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL);   /* Check status */
        2
        2       RETURN(NORMAL);                                 /* Return to caller */
        2       END PUT_BOX;
        1
        1       END DISP_TEMPLATE;
```

```
EXEC
V04-

980
980
980
980
980
980
980
980
980
980
1057
1057
1057
1057
1058
1058
1058
1058
1058
1058
1058
1058
1058
1058
1059
1059
1059
1059
1059
1059
1059
1059
1059
1059
1060
1060
1060
1060
1060
1060
1060
1060

COM
---

PLI/
```

```
            COLLECTION_END: Procedure Returns(fixed binary(31));     /* Indicate collection ended */
      1
  !   1     /*
  !   1     /*++
  !   1     /*
  !   1     /* FUNCTIONAL DESCRIPTION:
  !   1     /*
  !   1     /*      COLLECTION_END
  !   1     /*
  !   1     /*      Called by CTRLC, CTRLZ, COLLECTION_EVENT or CLASS_COLLECT whenever it
  !   1     /*      is determined that data collection has reached an end. This can occur
  !   1     /*      when the user strikes CTRL-C or CTRL-Z, an input (playback) file has
  !   1     /*      reached end-of-file, or a requested ending time has occurred.
  !   1     /*
  !   1     /* INPUTS:
  !   1     /*
  !   1     /*      None
  !   1     /*
  !   1     /* OUTPUTS:
  !   1     /*
  !   1     /*      None
  !   1     /*
  !   1     /* IMPLICIT OUTPUTS:
  !   1     /*
  !   1     /*      COLLENDED bit is set.
  !   1     /*
  !   1     /* ROUTINE VALUE:
  !   1     /*
  !   1     /*      SS$_NORMAL
  !   1     /*
  !   1     /* SIDE EFFECTS:
  !   1     /*
  !   1     /*      All timers are canceled, a $WAKE is issued, and the display and
  !   1     /*      "between screens" event flags are set. Also, I/O is canceled on the
  !   1     /*      channel for CTRL-C and CTRL-Z to disable reception of AST's, and
  !   1     /*      on the channel for CTRL-W.
  !   1     /*
  !   1     /*--
  !   1     /*/
      1
```

```
        /*
        /*      |-----------------------------------------------------------------------------+
        /*      |                                                                             |
        /*      |                              LOCAL STORAGE                                  |
        /*      |                                                                             |
        /*      |-----------------------------------------------------------------------------+
        /*/

        %INCLUDE        SYS$CANTIM;                                            /* $CANTIM system service */
        %INCLUDE        SYS$SETEF;                                             /* $SETEF system service */
        %INCLUDE        SYS$CANCEL;                                            /* $CANCEL system service */
        %INCLUDE        MONDEF;                                                /* Monitor utility structure definitions */

        Declare
          DISP_EV_FLAG  FIXED BINARY(31) GLOBALREF VALUE,                      /* Display event flag */
          BET_EV_FLAG   FIXED BINARY(31) GLOBALREF VALUE,                      /* "Between screens" display event flag */
          HIB_EV_FLAG   FIXED BINARY(31) GLOBALREF VALUE,                      /* Hibernation event flag */
          COLLENDED     BIT(1)           GLOBALREF,                            /* YES => collection has ended */
          MRBPTR        POINTER          GLOBALREF,                            /* Pointer to MRB (Monitor Request Block) */
          MCAPTR        POINTER          GLOBALREF,                            /* Pointer to MCA (Monitor Communication Area) */
          CTRLCZ_CHAN   FIXED BINARY(31) GLOBALREF,                            /* Channel number for CTRL-C and CTRL-Z */
          CTRLW_CHAN    FIXED BINARY(31) GLOBALREF,                            /* Channel number for CTRL-W */
          NORMAL        FIXED BINARY(31) GLOBALREF,                            /* MONITOR normal status value */
          CALL          FIXED BINARY(31);                                      /* Holds function value (return status) of called routines *

        CALL = SYS$CANTIM(,);                                                  /* Cancel outstanding timer requests */
        CALL = SYS$SETEF(HIB_EV_FLAG);                                         /* Wake up if hibernating for future request */
        CALL = SYS$SETEF(DISP_EV_FLAG);                                        /* Force final display */
        CALL = SYS$SETEF(BET_EV_FLAG);                                         /* Force final screen of multi-screen display */
        COLLENDED = YES;                                                       /* Indicate collection ended */
        MRBPTR->MRB$Q_ENDING = MCAPTR->MCA$Q_LASTCOLL;                         /* Establish last collection time as ending */
        IF CTRLCZ_CHAN ^= 0 THEN CALL = SYS$CANCEL(CTRLCZ_CHAN); /* Cancel CTRL-C and CTRL-Z handlers */
        CTRLCZ_CHAN = 0;                                                       /* ... and indicate so */
        IF CTRLW_CHAN ^= 0 THEN CALL = SYS$CANCEL(CTRLW_CHAN);                 /* Cancel CTRL-W handler */
        CTRLW_CHAN = 0;                                                        /* ... and indicate so */
        RETURN(NORMAL);                                                        /* Return to caller */

        END COLLECTION_END;
```

```
CTRLC: Procedure Returns(fixed binary(31));                    /* CTRL-C handler */

/*
/*++
/*
/* FUNCTIONAL DESCRIPTION:
/*
/*      CTRLC
/*
/*      AST Routine entered whenever the user strikes CTRL-C.
/*      The COLLECTION_END routine is called to begin termination
/*      of the Monitor request. Also, the CTRLCZ_HIT bit is set,
/*      and the PROMPT bit is set to indicate a MONITOR> prompt
/*      is desired.
/*
/* INPUTS:
/*
/*      None
/*
/* OUTPUTS:
/*
/*      None
/*
/* ROUTINE VALUE:
/*
/*      SS$_NORMAL
/*
/*--
/*/

/*
/*      !-----------------------------------------------------------------+
/*      !                                                                 !
/*      !                       LOCAL STORAGE                             !
/*      !                                                                 !
/*      !-----------------------------------------------------------------+
/*/

Declare
  COLLECTION_END ENTRY,                                    /* Routine to indicate end of collection */
  CTRLCZ_HIT     BIT(1) ALIGNED  GLOBALREF,                /* YES => CTRL-C or CTRL-Z has been hit */
  PROMPT         BIT(1) ALIGNED  GLOBALREF,                /* YES => prompt user for another subcommand */
  NORMAL         FIXED BINARY(31) GLOBALREF;               /* MONITOR normal status value */

CTRLCZ_HIT = YES;                                          /* Indicate CTRL-C has been hit */
PROMPT = YES;                                              /* Indicate user wants MONITOR> prompt */
CALL COLLECTION_END();                                     /* Indicate end of collection */
RETURN(NORMAL);                                            /* Return to caller */

END CTRLC;
```

```
        CTRLZ: Procedure Returns(fixed binary(31));                   /* CTRL-Z handler */
   1
   1    /*
   1    /*++
   1    /*
   1    /* FUNCTIONAL DESCRIPTION:
   1    /*
   1    /*      CTRLZ
   1    /*
   1    /*      AST Routine entered whenever the user strikes CTRL-Z.
   1    /*      The COLLECTION_END routine is called to begin termination
   1    /*      of the Monitor request. Also, the CTRLCZ_HIT bit is set,
   1    /*      and the PROMPT bit is set to 0 to indicate a MONITOR> prompt
   1    /*      is NOT desired.
   1    /*
   1    /* INPUTS:
   1    /*
   1    /*      None
   1    /*
   1    /* OUTPUTS:
   1    /*
   1    /*      None
   1    /*
   1    /* ROUTINE VALUE:
   1    /*
   1    /*      SS$_NORMAL
   1    /*
   1    /*--
   1    /*/
   1
   1    /*
   1    /*      +--------------------------------------------------------------+
   1    /*      |                                                              |
   1    /*      |                        LOCAL STORAGE                         |
   1    /*      |                                                              |
   1    /*      +--------------------------------------------------------------+
   1    /*/
   1
   1    Declare
   1      COLLECTION_END ENTRY,                                       /* Routine to indicate end of collection */
   1      COMMAND_FILE  FILE              GLOBALREF,                  /* File reference for the execute command file */
   1      CTRLZ_HIT     BIT(1) ALIGNED    GLOBALREF,                  /* YES => CTRL-Z has been hit */
   1      CTRLCZ_HIT    BIT(1) ALIGNED    GLOBALREF,                  /* YES => CTRL-C or CTRL-Z has been hit */
   1      PROMPT        BIT(1) ALIGNED    GLOBALREF,                  /* YES => prompt user for another subcommand */
   1      EXECUTE       BIT(1) ALIGNED    GLOBALREF,                  /* YES => read another execute command file subcommand */
   1      NORMAL        FIXED BINARY(31) GLOBALREF;                   /* MONITOR normal status value */
   1
   1    CTRLZ_HIT = YES;                                              /* Indicate CTRL-Z has been hit */
   1    CTRLCZ_HIT = YES;                                             /* Indicate CTRL-Z has been hit */
   1    PROMPT = NO;                                                  /* Indicate user does NOT want a MONITOR> prompt */
   1    IF (EXECUTE = YES) THEN DO;                                   /* If there is an execute command file open, */
   2        CLOSE FILE(COMMAND_FILE);                                 /*  close the execute command file and */
   2        EXECUTE = NO;                                             /*  indicate no more execute subcommands to be done. */
   2        END;
   1    CALL COLLECTION_END();                                        /* Indicate end of collection */
   1    RETURN(NORMAL);                                               /* Return to caller */
```

```
        1
        1      END CTRLZ;

```

```
CTRLW: Procedure Returns(fixed binary(31));                 /* CTRL-W (display screen refresh) handler */

/*
/*+++
/*
/* FUNCTIONAL DESCRIPTION:
/*
/*       CTRLW
/*
/*       AST routine, entered whenever the user strikes CTRL-W.
/*       Sets the Refresh Event Flag to indicate a new display
/*       event (including template) is desired.
/*
/* INPUTS:
/*
/*       None
/*
/* OUTPUTS:
/*
/*       None
/*
/* ROUTINE VALUE:
/*
/*       SS$_NORMAL
/*
/*--
/*/

/*
/*       |-------------------------------------------------------------+
/*       |                                                             |
/*       |                        LOCAL STORAGE                        |
/*       |                                                             |
/*       |-------------------------------------------------------------+
/*/

%INCLUDE        SYS$SETEF;                                  /* $SETEF system service */

Declare
  REFR_EV_FLAG  FIXED BINARY(31) GLOBALREF VALUE,          /* Refresh event flag */
  NORMAL        FIXED BINARY(31) GLOBALREF,                /* MONITOR normal status value */
  CALL          FIXED BINARY(31);                          /* Holds function value (return status) of called routines *

CALL = SYS$SETEF(REFR_EV_FLAG);                             /* Cause refresh display event to occur */
RETURN(NORMAL);                                             /* Return to caller */

END CTRLW;
```

```
            WRITE_HEADER: Procedure Returns(fixed binary(31));                  /* Write recording file header record */
  :    1                                                                         /* ... and system information record */
       1
  !    1    /*
  !    1    /*++
  !    1    /*
  !    1    /* FUNCTIONAL DESCRIPTION:
  !    1    /*
  !    1    /*      WRITE_HEADER
  !    1    /*
  !    1    /*      Called by the CLASS_COLLECT routine to write the first 2
  !    1    /*      records of the recording file (File Header Record and
  !    1    /*      System Information Record). Called once per Monitor
  !    1    /*      request before any class records are written.
  !    1    /*
  !    1    /* INPUTS:
  !    1    /*
  !    1    /*      None
  !    1    /*
  !    1    /* OUTPUTS:
  !    1    /*
  !    1    /*      None
  !    1    /*
  !    1    /* ROUTINE VALUE:
  !    1    /*
  !    1    /*      SS$_NORMAL, or failing MONITOR status code.
  !    1    /*
  !    1    /*--
  !    1    /*/
       1
  !    1    /*
  !    1    /*      !-------------------------------------------------------------+
  !    1    /*      !                                                             !
  !    1    /*      !                        LOCAL STORAGE                        !
  !    1    /*      !                                                             !
  !    1    /*      !-------------------------------------------------------------+
  !    1    /*/
       1
       1    %INCLUDE        MONDEF;                                             /* Monitor utility structure definitions */
       1
       1    Declare
       1      WRITE_RECORD  ENTRY (ANY) RETURNS(FIXED BINARY(31));              /* Routine to write a rec to the recording file */
       1
       1    Declare
       1      CALL          FIXED BINARY(31),                                   /* Holds function value (return status) of called ro
       1      STATUS        BIT(1)  BASED(ADDR(CALL)),                          /* Low-order status bit for called routines */
       1      NORMAL        FIXED BINARY(31) GLOBALREF,                         /* MONITOR normal status value */
       1      SPTR          POINTER GLOBALREF,                                  /* Pointer to SYI (System Information Area) */
       1      MRBPTR        POINTER GLOBALREF,                                  /* Pointer to MRB (Monitor Request Block) */
       1      M             POINTER DEFINED(MRBPTR),                            /* Synonym for MRBPTR */
       1      H             POINTER,                                            /* Pointer to record file header */
       1      HEADER_TYPE   FIXED BINARY(15) GLOBALREF,                         /* Type for MONITOR recording file header */
       1      ST_LEVEL_CUR  CHAR(8) GLOBALREF,                                  /* Current MONITOR recording file structure level */
       1      ST_LEVEL_PB   CHAR(8) GLOBALREF,                                  /* MONITOR recording file structure level from input
       1      REVLEVELS     CHAR(128) GLOBALREF,                                /* Revision levels used by classes for this request
       1      REV0CLSBITS   BIT(128) GLOBALREF,                                 /* Bits for classes recorded at rev level 0 */
```

```
        1       1 COMM_D BASED(M->MRB$A_COMMENT),                              /* Descriptor for user's comment string */
        1         2 L           FIXED BINARY(15),                              /* Length */
        1         2 TC          CHAR(2),                                       /* Type and class */
        1         2 A           POINTER,                                       /* Address */
        1       COMMENT         CHAR(COMM_D.L) BASED(COMM_D.A),                /* User-specified comment string */
        1       1 REC_DESCR,                                                   /* Record descriptor */
        1         2 L           FIXED BINARY(31),                              /* Length */
        1         2 A           POINTER;                                       /* Address */
        1
```

```
     1      ALLOCATE FILE_HDR SET (H);                         /* Allocate file header space */
     1
     1      H->MNR_HDR$B_TYPE = UNSPEC(HEADER_TYPE);           /* Load header type code */
     1      H->MNR_HDR$V_FILLER = '0'B;                        /* Clear all unused flags */
     1      H->MNR_HDR$Q_BEGINNING = M->MRB$Q_BEGINNING;       /* Load beginning time */
     1      H->MNR_HDR$Q_ENDING = '0'B;                        /* Indicate no ending time yet */
     1      H->MNR_HDR$L_INTERVAL = M->MRB$L_INTERVAL;         /* Load interval value */
     1      H->MNR_HDR$O_REV0CLSBITS = REV0CLSBITS;            /* Load bits for classes recorded at rev level 0 */
     1      H->MNR_HDR$L_RECCT = 0;                            /* Indicate no records yet */
     1      IF M->MRB$V_PLAYBACK                               /* If a playback request, */
     1           THEN H->MNR_HDR$T_LEVEL = ST_LEVEL_PB;        /*    then load playback recording file structure level */
     1           ELSE H->MNR_HDR$T_LEVEL = ST_LEVEL_CUR;       /*    else load current recording file structure level */
     1
     1      IF M->MRB$A_COMMENT = NULL()                       /* If no comment string specified, */
     1           THEN DO;
     2                H->MNR_HDR$T_COMMENT = ' ';              /* Load a string of blanks */
     2                H->MNR_HDR$W_COMLEN = 0;                 /* ... and a length of 0 */
     2                END;
     1
     1           ELSE DO;                                      /* Comment string is specified */
     2                H->MNR_HDR$T_COMMENT = COMMENT;          /* Load user's comment string */
     2                H->MNR_HDR$W_COMLEN = COMM_D.L;          /* ... and its actual length */
     2                IF H->MNR_HDR$W_COMLEN > MNR_HDR$K_MAXCOMLEN         /* Minimize actual length with ... */
     2                     THEN H->MNR_HDR$W_COMLEN = MNR_HDR$K_MAXCOMLEN; /* ... max comment length */
     2                END;
     1
     1      H->MNR_HDR$O_CLASSBITS = M->MRB$O_CLASSBITS;       /* Load class bit string */
     1      H->MNR_HDR$T_REVLEVELS = REVLEVELS;                /* Load revision levels used by this request */
     1
 !   1      /*
 !   1      /*        Write file header record
 !   1      /*/
     1
     1      REC_DESCR.L = MNR_HDR$K_SIZE;                      /* Load up length */
     1      REC_DESCR.A = H;                                   /* ... and address of record for write */
     1      CALL = WRITE_RECORD(REC_DESCR);                    /* Write the file header record */
     1      FREE H->FILE_HDR;                                  /* Free file header space */
     1      IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL);      /* Check WRITE_RECORD call */
     1
 !   1      /*
 !   1      /*        Write system information record
 !   1      /*/
     1
     1      REC_DESCR.L = MNR_SYI$K_SIZE;                      /* Load up length */
     1      REC_DESCR.A = SPTR;                                /* ... and address of record for write */
     1      CALL = WRITE_RECORD(REC_DESCR);                    /* Write the system info record */
     1      IF STATUS = NOT_SUCCESSFUL THEN RETURN(CALL);      /* Check WRITE_RECORD call */
     1
     1      RETURN(NORMAL);                                    /* Return to caller */
     1      END WRITE_HEADER;
```

```
          WRITE_RECORD: Procedure (RECORD_DESC)
                                  Returns(fixed binary(31));
     1
:    1    /*
:    1    /*++
:    1    /*
:    1    /* FUNCTIONAL DESCRIPTION:
:    1    /*
:    1    /*      WRITE_RECORD
:    1    /*
:    1    /*      Called by the WRITE_HEADER and CLASS_COLLECT routines to
:    1    /*      write a single record to the recording file. If a flush
:    1    /*      has been indicated, it is performed.
:    1    /*
:    1    /* INPUTS:
:    1    /*
:    1    /*      Address of a string descriptor describing the record to be written.
:    1    /*
:    1    /* IMPLICIT INPUTS:
:    1    /*
:    1    /*      FLUSH_IND -- Flush indicator. If set, perform an RMS flush operation
:    1    /*                   to "checkpoint" the recording file.
:    1    /*
:    1    /* OUTPUTS:
:    1    /*
:    1    /*      None
:    1    /*
:    1    /* IMPLICIT OUTPUTS:
:    1    /*
:    1    /*      RECCT incremented by 1.
:    1    /*
:    1    /* ROUTINE VALUE:
:    1    /*
:    1    /*      SS$_NORMAL
:    1    /*
:    1    /*--
:    1    /*/
     1
```

```
 ¦  1      /*
 ¦  1      /*        +-----------------------------------------------------------------+
 ¦  1      /*        ¦                                                                 ¦
 ¦  1      /*        ¦                        LOCAL STORAGE                            ¦
 ¦  1      /*        ¦                                                                 ¦
 ¦  1      /*        +-----------------------------------------------------------------+
 ¦  1      /*/
    1
    1      Declare
    1        NORMAL                FIXED BINARY(31) GLOBALREF,            /* MONITOR normal status value */
    1        RECCT                 FIXED BINARY(31) GLOBALREF,            /* Count of records written to record file */
    1        FLUSH_IND             BIT(1) ALIGNED   GLOBALREF;            /* Flush indicator; YES => perform FLUSH */
    1
    1      Declare
    1        1 RECORD_DESC,                                               /* Record descriptor */
    1          2 RECORD_LEN        FIXED BINARY(31),                      /* Record length */
    1          2 RECORD_PTR        POINTER,                               /* Record pointer */
    1        RECORD_DATA           CHAR(RECORD_LEN) BASED(RECORD_PTR);    /* Record data */
    1
    1      Declare
    1        RECORD_FILE           FILE RECORD;                           /* Monitor Record File */
    1
    1      WRITE FILE(RECORD_FILE) FROM(RECORD_DATA);                     /* Write a record */
    1      RECCT = RECCT + 1;                                             /* Count it */
    1
    1      IF FLUSH_IND                                                   /* If flush indicated for this write, */
    1          THEN DO;
    2              CALL FLUSH(RECORD_FILE);                               /* Flush record file to checkpoint collected data */
    2              FLUSH_IND = NO;                                        /* Indicate flush not required */
    2              END;
    1
    1      RETURN(NORMAL);                                                /* Return */
    1      END WRITE_RECORD;

```

SHO
V04

63

63

```
READ_INPUT: Procedure (SKIP_IND);                                    /* Routine to read a record from the /INPUT file */

/*++
/*
/* FUNCTIONAL DESCRIPTION:
/*
/*      READ_INPUT
/*
/*      This routine reads the /INPUT (playback) file until a
/*      record of the desired type is found, or until end-of-file
/*      is reached. The following categories of record types exist:
/*
/*              Types   0 - 127:        Class record
/*              Types 128 - 191:        DIGITAL control record
/*              Types 192 - 255:        Customer control record
/*
/*      A class record is always desired. A customer control record
/*      is never desired. A DIGITAL control record can be desired
/*      or not, depending on the input parameter SKIP_IND.
/*
/* INPUTS:
/*
/*      SKIP_IND -- a binary longword value indicating whether or not
/*                      to skip past DIGITAL control records. If
/*                      SKIP_IND is 0, DIGITAL control records
/*                      are desired, and will not be skipped.
/*                      Otherwise, they are skipped.
/*
/* IMPLICIT INPUTS:
/*
/*      MCAPTR -- Pointer to Monitor Communication Area
/*      INPUT_CPTR -- Pointer to /INPUT file buffer
/*
/* OUTPUTS:
/*
/*      None
/*
/* IMPLICIT OUTPUTS:
/*
/*      MCA$L_INPUT_LEN is updated to indicate the length of the record
/*      currently in the input buffer.
/*
/*      MCA$V_EOF is set if end-of-file is reached.
/*
/* ROUTINE VALUE:
/*
/*      None
/*
/* SIDE EFFECTS:
/*
/*      /INPUT file (INPUT_FILE) is advanced to the desired record.
/*
/*/
```

```
 |   1      /*
 |   1      /*        |--------------------------------------------------------------------+
 |   1      /*        |                                                                    |
 |   1      /*        |                          LOCAL STORAGE                             |
 |   1      /*        |                                                                    |
 |   1      /*        |--------------------------------------------------------------------+
 |   1      /*/
     1
     1      %INCLUDE         MONDEF;                                              /* Monitor utility structure definitions */
     1
     1      Declare
     1        MAX_REC_SIZE   FIXED BINARY(31) GLOBALREF VALUE,                    /* Max record size for PLAYBACK & RECORD files */
     1        MCAPTR         POINTER GLOBALREF,                                   /* Pointer to MCA (Monitor Communication Area) */
     1        MC             POINTER DEFINED(MCAPTR),                             /* Synonym for MCAPTR */
     1        INPUT_CPTR     POINTER GLOBALREF,                                   /* Ptr to input buffer count word */
     1        INPUT_DATA     CHAR(MAX_REC_SIZE) VARYING BASED(INPUT_CPTR);        /* Playback file input buffer */
     1
     1      Declare
     1        SKIP_IND               FIXED BINARY(31),                            /* Skip indicator; non-zero => skip DIGITAL control
     1        DESIRED_TYPE           BIT(1) ALIGNED,                              /* YES => desired record type found */
     1        1 RECORD_TYPE          BASED(MC->MCA$A_INPUT_PTR),                  /* Record type field of input record */
     1          2 FILLER             BIT(6),
     1          2 BIT6               BIT(1),
     1          2 BIT7               BIT(1);
     1
     1      Declare
     1        INPUT_FILE             FILE RECORD INPUT;                           /* Monitor Input (Playback) File */
     1
     1      DESIRED_TYPE = NO;                                                    /* Don't have desired type yet */
     1      DO WHILE (^ MC->MCA$V_EOF & ^ DESIRED_TYPE);                          /* Stop reading when hit EOF or desired rec found */
     2      READ FILE(INPUT_FILE) INTO(INPUT_DATA);                               /* Read a record from the input file */
     2      IF BIT7 = NO                                                          /* If high-order bit of record type off, */
     2          THEN DESIRED_TYPE = YES;                                          /*    then we found a desired type (class record) */
     2          ELSE IF SKIP_IND = 0 & BIT6 = NO                                  /* If caller wants a DIGITAL control rec, */
     2               THEN DESIRED_TYPE = YES;                                     /*    and it is present, let him have it */
     2      END;
     1
     1      MC->MCA$L_INPUT_LEN = LENGTH(INPUT_DATA);                             /* Establish length of input */
     1
     1      RETURN;                                                               /* Return */
     1      END READ_INPUT;
```

COMMAND LINE
------- ----

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

F 13

SHO
V04

63

63

63

63

63

0242 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

MONMSG
LIS

REQUEST
LIS

SHODEF
LIS

MONSUB
LIS

PREPOST
LIS

SUMMBUFF
LIS