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

```
MM      MM    000000    NN      NN  MM      MM    AAAAAA     IIIIII    NN      NN
MM      MM    000000    NN      NN  MM      MM    AAAAAA     IIIIII    NN      NN
MMMM  MMMM  00      00  NN      NN  MMMM  MMMM  AA      AA     II      NN      NN
MMMM  MMMM  00      00  NN      NN  MMMM  MMMM  AA      AA     II      NN      NN
MM  MM  MM  00      00  NNNN    NN  MM  MM  MM  AA      AA     II      NNNN    NN
MM  MM  MM  00      00  NNNN    NN  MM  MM  MM  AA      AA     II      NNNN    NN
MM      MM  00      00  NN  NN  NN  MM      MM  AA      AA     II      NN  NN  NN
MM      MM  00      00  NN  NN  NN  MM      MM  AA      AA     II      NN  NN  NN
MM      MM  00      00  NN    NNNN  MM      MM  AAAAAAAAAA     II      NN    NNNN
MM      MM  00      00  NN    NNNN  MM      MM  AAAAAAAAAA     II      NN    NNNN
MM      MM  00      00  NN      NN  MM      MM  AA      AA     II      NN      NN      ....
MM      MM  00      00  NN      NN  MM      MM  AA      AA     II      NN      NN      ....
MM      MM    000000    NN      NN  MM      MM  AA      AA   IIIIII    NN      NN      ....
MM      MM    000000    NN      NN  MM      MM  AA      AA   IIIIII    NN      NN      ....

LL              IIIIII      SSSSSSSS
LL              IIIIII      SSSSSSSS
LL                II      SS
LL                II      SS
LL                II      SS
LL                II      SS
LL                II        SSSSSS
LL                II        SSSSSS
LL                II              SS
LL                II              SS
LL                II              SS
LL                II              SS
LLLLLLLLLL      IIIIII    SSSSSSSS
LLLLLLLLLL      IIIIII    SSSSSSSS
```

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

```
46  1       /*
47  1       /* MODIFIED BY:
48  1       /*
49  1       /*      V03-018 TLC1089         Thomas L. Cafarella     26-Jul-1984     11:00
50  1       /*              Accept a space character in time specification.
51  1       /*
52  1       /*      V03-017 TLC1087         Thomas L. Cafarella     25-Jul-1984     15:00
53  1       /*              Default to /ALL when summarizing.
54  1       /*
55  1       /*      V03-016 TLC1075         Thomas L. Cafarella     27-Jun-1984     15:00
56  1       /*              Add stickiness to /INPUT qualifier.
57  1       /*
58  1       /*      V03-015 TLC1073         Thomas L. Cafarella     02-May-1984     13:00
59  1       /*              Make MAX_INP_FILES limit bigger.
60  1       /*
61  1       /*      V03-014 PRS1012         Paul R. Senn            23-Mar-1984     14:00
62  1       /*              Add wildcard capability for MF summary.
63  1       /*
64  1       /*      V03-013 TLC1056         Thomas L. Cafarella     23-Mar-1984     13:00
65  1       /*              Exclude class which is disabled.
66  1       /*
67  1       /*      V03-012 PRS1011         Paul R. Senn            29-Feb-1984     14:00
68  1       /*              add /FLUSH_INTERVAL qualifier
69  1       /*
70  1       /*      V03-011 TLC1052         Thomas L. Cafarella     17-Feb-1984     11:00
71  1       /*              Add multi-file summary capability.
72  1       /*
73  1       /*      V03-010 PRS1002         Paul R. Senn            29-Dec-1983     16:00
74  1       /*              GLOBALDEF VALUE symbols must now be longwords;
75  1       /*              Use %REPLACE rather than GLOBALDEF VALUE for any equated
76  1       /*              symbols which are not 4 bytes in length;
77  1       /*
78  1       /*      V03-010 PRS1001         Paul R. Senn            27-Dec-1983     16:00
79  1       /*              Add ALL CLASSES Pseudo-class
80  1       /*
81  1       /*      V03-009 TLC1044         Thomas L. Cafarella     24-Aug-1983     13:00
82  1       /*              Eliminate CLI "NOCOMD" error for comment lines.
83  1       /*
84  1       /*      V03-008 SPC0007         Stephen P. Carney       24-Jun-1983     16:00
85  1       /*              Add EXECUTE subcommand.
86  1       /*
87  1       /*      V03-007 TLC1042         Thomas L. Cafarella     19-Jun-1983     15:00
88  1       /*              Add /ITEM qualifier for homogeneous classes.
89  1       /*
90  1       /*      V03-007 TLC1041         Thomas L. Cafarella     16-Jun-1983     15:00
91  1       /*              Ignore CLI error message when no command on line.
92  1       /*
93  1       /*      V03-007 TLC1038         Thomas L. Cafarella     14-Jun-1983     18:00
94  1       /*              Make default list of classes replace previous list.
95  1       /*
96  1       /*      V03-006 TLC1028         Thomas L. Cafarella     14-Apr-1983     16:00
97  1       /*              Add interactive user interface.
98  1       /*
99  1       /*      V03-005 TLC1019         Thomas L. Cafarella     18-Jun-1982     16:00
100 1       /*              Change CLI$_NEGATED symbol to CLI$_LOCNEG.
101 1       /*
```

```
  102  :  1      /*      V03-004 TLC1012         Thomas L. Cafarella     30-Mar-1982     13:00
  103  :  1      /*              Display user's comment string on screen line 5.
  104  :  1      /*
  105  :  1      /*      V03-004 TLC1011         Thomas L. Cafarella     29-Mar-1982     20:00
  106  :  1      /*              Move system service names for SSERROR msg to static storage.
  107  :  1      /*
  108  :  1      /*      V03-003 TLC1009         Thomas L. Cafarella     29-Mar-1982     01:00
  109  :  1      /*              Get current time when other times are converted.
  110  :  1      /*
  111  :  1      /*      V03-003 TLC1007         Thomas L. Cafarella     28-Mar-1982     19:00
  112  :  1      /*              Add checks for maximum sizes of qualifier values.
  113  :  1      /*
  114  :  1      /*      V03-002 TLC1003         Thomas L. Cafarella     23-Mar-1982     13:00
  115  :  1      /*              Fix up module headers.
  116  :  1      /*
  117  :  1      /*      V03-001 TLC1001         Thomas L. Cafarella     16-Mar-1982     13:00
  118  :  1      /*              Add CTRL-W screen refresh support.
  119  :  1      /*
  120  :  1      /*--
  121  :  1      /*/
  122     1
```

```
123  1      /*
124  1      /*      !-----------------------------------------------------------------------+
125  1      /*      !                                                                       !
126  1      /*      !                          INCLUDE  FILES                               !
127  1      /*      !                                                                       !
128  1      /*      !-----------------------------------------------------------------------+
129  1      /*/
130  1
131  1      %INCLUDE        MONDEF;                         /* Monitor utility structure definitions */
899  1      %INCLUDE        $CHFDEF;                        /* Condition handler facility definitions */
919  1      %INCLUDE        $STSDEF;                        /* Status code definitions */
936  1
937  1      /*
938  1      /*      !-----------------------------------------------------------------------+
939  1      /*      !                                                                       !
940  1      /*      !                  SYSTEM SERVICE MACRO DEFINITIONS                     !
941  1      /*      !                                                                       !
942  1      /*      !-----------------------------------------------------------------------+
943  1      /*/
944  1
945  1
```

```
946  1     /*
947  1     /*      +-----------------------------------------------------------------------+
948  1     /*      |                                                                       |
949  1     /*      |                        COMPILE-TIME CONSTANTS                         |
950  1     /*      |                                                                       |
951  1     /*      +-----------------------------------------------------------------------+
952  1     /*/
953  1
954  1     %REPLACE NOT_SUCCESSFUL         BY '0'B;            /* Failing status bit */
955  1     %REPLACE YES                    BY '1'B;            /* For general use */
956  1     %REPLACE NO                     BY '0'B;            /* For general use */
957  1
```

```
958  1       /*
959  1       /*      +-----------------------------------------------------------------------+
960  1       /*      |                                                                       |
961  1       /*      |                    EXTERNAL STORAGE  DEFINITIONS                      |
962  1       /*      |                                                                       |
963  1       /*      +-----------------------------------------------------------------------+
964  1       /*/
965  1
966  1       Declare
967  1       MON_ERR          ENTRY (ANY VALUE, ANY, ANY) OPTIONS(VARIABLE),
968  1                                                          /* Routine to log synchronous errors */
969  1       SIGNAL_MON_ERR   ENTRY,                            /* Routine to signal MONITOR errors */
970  1       DISPLAY_CLEANUP  ENTRY RETURNS(FIXED BINARY(31)),  /* Procedure clean up DISPLAY processing */
971  1       SS$_NORMAL       FIXED BINARY(31) GLOBALREF VALUE, /* System normal return status */
972  1       RMS$_EOF         FIXED BINARY(31) GLOBALREF VALUE, /* RMS end-of-file return status */
973  1       RMS$_NMF         FIXED BINARY(31) GLOBALREF VALUE, /* RMS no-more-files message for wildcard parsing */
974  1       RMS$_FNF         FIXED BINARY(31) GLOBALREF VALUE, /* RMS file-not-found message */
975  1       NORMAL           FIXED BINARY(31) GLOBALDEF;       /* MONITOR normal return status */
976  1
```

```
  977     1       Declare
  978     1       MNR$_ERREXEREA  FIXED BINARY(31) GLOBALREF VALUE,      /* Error message code */
  979     1       MNR$_ERRRECFIL  FIXED BINARY(31) GLOBALREF VALUE,      /* Error message code */
  980     1       MNR$_UNEXPERR   FIXED BINARY(31) GLOBALREF VALUE,      /* Error message code */
  981     1       MNR$_ERRPARSE   FIXED BINARY(31) GLOBALREF VALUE,      /* Error message code */
  982     1       CLI$_NOCOMD     FIXED BINARY(31) GLOBALREF VALUE;      /* CLI error message code */
  983     1
```

```
 984 :  1      /*
 985 :  1      /*      !-----------------------------------------------------------------------+
 986 :  1      /*      !                                                                       !
 987 :  1      /*      !                       GLOBAL STORAGE DEFINITIONS                      !
 988 :  1      /*      !                                                                       !
 989 :  1      /*      !-----------------------------------------------------------------------+
 990 :  1      /*/
 991    1
 992    1      Declare
 993    1      COMMAND_FILE            FILE            GLOBALREF;              /* Execute Command File */
 994    1
 995    1      Declare
 996    1      CURR_ERRCODE            FIXED BINARY(31) GLOBALDEF INIT(0),     /* MONITOR error status code currently expected */
 997    1      FIRST_MON_CMD           BIT(1) ALIGNED GLOBALDEF INIT('0'B),    /* YES => first MONITOR (DCL-level) cmd executing */
 998    1      PROMPT                  BIT(1) ALIGNED GLOBALDEF INIT('0'B),    /* YES => prompt user for another subcommand */
 999    1      EXECUTE                 BIT(1) ALIGNED GLOBALDEF INIT('0'B),    /* YES => read another command from the execute file
1000    1      DISPLAYING              BIT(1) ALIGNED GLOBALDEF INIT('0'B);    /* YES => terminal display output is active */
1001    1
1002    1      Declare
1003    1      CDBPTR                  POINTER GLOBALDEF,              /* Pointer to CDB (Class Descriptor Block) */
1004    1      MRBPTR                  POINTER GLOBALDEF,              /* Pointer to MRB (Monitor Request Block) */
1005    1      DEF_MRBPTR              POINTER GLOBALDEF,              /* Pointer to "default" MRB (Monitor Request Block) */
1006    1      TEMP_MRBPTR             POINTER GLOBALDEF,              /* Pointer to "temporary" MRB (Monitor Request Block) */
1007    1      CURR_MRBPTR             POINTER GLOBALDEF,              /* Pointer to "current" MRB (Monitor Request Block) */
1008    1      ACT_MRBPTR              POINTER GLOBALDEF,              /* Pointer to "active" MRB (Monitor Request Block) */
1009    1      MCAPTR                  POINTER GLOBALDEF,              /* Pointer to MCA (Monitor Communication Area) */
1010    1      SPTR                    POINTER GLOBALDEF;              /* Pointer to SYI (System Information Area) */
1011    1
1012    1      Declare
1013    1      QUALPTR                 POINTER GLOBALDEF,              /* Pointer to Qualifier Descriptors Block */
1014    1      DEFPTR                  POINTER GLOBALDEF;              /* Pointer to Qualifier Default Value Descriptors Block */
1015    1
1016    1      Declare
1017    1      1 STAT_TABLE    GLOBALDEF,                              /* Table of pointers to str descrs for statistic qualifiers
1018    1        2 STAT_DESC   (0:STATS-1) POINTER;
1019    1
1020    1      Declare
1021    1      1 PROCD_TABLE   GLOBALDEF,                              /* Table of pointers to str descrs for PROCESSES display qua
1022    1        2 PROCD_DESC  (0:PROCDISPS-1) POINTER;
1023    1
```

```
1024    1
1025    1       /*
1026    1       /*      Counted strings for system service names used in the MNR$_SSERROR error message
1027    1       /*/
1028    1
1029    1       Declare
1030    1         1 READEF_STR GLOBALDEF,                       /* Counted string for $READEF */
1031    1                 2 L           FIXED BINARY(7) INIT(7),  /* Length */
1032    1                 2 S           CHAR(7) INIT('$READEF'),  /* String */
1033    1
1034    1         1 CLREF_STR GLOBALDEF,                        /* Counted string for $CLREF */
1035    1                 2 L           FIXED BINARY(7) INIT(6),  /* Length */
1036    1                 2 S           CHAR(7) INIT('$CLREF'),   /* String */
1037    1
1038    1         1 SCHDWK_STR GLOBALDEF,                       /* Counted string for $SCHDWK */
1039    1                 2 L           FIXED BINARY(7) INIT(7),  /* Length */
1040    1                 2 S           CHAR(7) INIT('$SCHDWK'),  /* String */
1041    1
1042    1         1 SETIMR_STR GLOBALDEF,                       /* Counted string for $SETIMR */
1043    1                 2 L           FIXED BINARY(7) INIT(7),  /* Length */
1044    1                 2 S           CHAR(7) INIT('$SETIMR'),  /* String */
1045    1
1046    1         1 DCLAST_STR GLOBALDEF,                       /* Counted string for $DCLAST*/
1047    1                 2 L           FIXED BINARY(7) INIT(7),  /* Length */
1048    1                 2 S           CHAR(7) INIT('$DCLAST');  /* String */
1049    1
```

```
1050 :  1       /*
1051 :  1       /*      +-----------------------------------------------------------------------+
1052 :  1       /*      |                                                                       |
1053 :  1       /*      |                              OWN STORAGE                              |
1054 :  1       /*      |                                                                       |
1055 :  1       /*      +-----------------------------------------------------------------------+
1056 :  1       /*/
1057    1
1058    1       Declare
1059    1       CALL                    FIXED BINARY(31),               /* Holds function value (return status) of called routines *
1060    1       STATUS                  BIT(1)  BASED(ADDR(CALL));      /* Low-order status bit for called routines */
1061    1
1062 :  1       /*
1063 :  1       /*      Strings for command qualifiers and parameter. Descriptors
1064 :  1       /*      for these strings are defined in the QUAL structure.
1065 :  1       /*/
1066    1
1067    1       DECLARE
1068    1
1069    1       BEG_QUAL_S      CHAR(9) INIT('BEGINNING'),                      /* BEGINNING qualifier string */
1070    1       END_QUAL_S      CHAR(6) INIT('ENDING'),                         /* ENDING qualifier string */
1071    1       INT_QUAL_S      CHAR(8) INIT('INTERVAL'),                       /* INTERVAL qualifier string */
1072    1       FLUSH_QUAL_S    CHAR(14) INIT('FLUSH_INTERVAL'),                /* FLUSH_INTERVAL qualifier string */
1073    1       VIEW_QUAL_S     CHAR(12) INIT('VIEWING_TIME'),                  /* VIEWING_TIME qualifier string */
1074    1       INP_QUAL_S      CHAR(5) INIT('INPUT'),                          /* INPUT qualifier string */
1075    1       DISP_QUAL_S     CHAR(7) INIT('DISPLAY'),                        /* DISPLAY qualifier string */
1076    1       REC_QUAL_S      CHAR(6) INIT('RECORD'),                         /* RECORD qualifier string */
1077    1       SUMM_QUAL_S     CHAR(7) INIT('SUMMARY'),                        /* SUMMARY qualifier string */
1078    1       COMM_QUAL_S     CHAR(7) INIT('COMMENT'),                        /* COMMENT qualifier string */
1079    1       BY_NODE_QUAL_S  CHAR(7) INIT('BY_NODE'),                        /* BY_NODE qualifier string */
1080    1       CLASS_PARM_S    CHAR(10) INIT('CLASS_NAME');                    /* CLASS_NAME parameter string */
1081    1
1082 :  1       /*
1083 :  1       /*      Strings for class-name qualifiers. Descriptors for these strings are defined in
1084 :  1       /*      the QUAL structure.
1085 :  1       /*/
1086    1
1087    1       DECLARE
1088    1
1089    1       ALL_QUAL_S      CHAR(3) INIT('ALL'),                            /* ALL qualifier string */
1090    1       CUR_QUAL_S      CHAR(7) INIT('CURRENT'),                        /* CURRENT qualifier string */
1091    1       AVE_QUAL_S      CHAR(7) INIT('AVERAGE'),                        /* AVERAGE qualifier string */
1092    1       MIN_QUAL_S      CHAR(7) INIT('MINIMUM'),                        /* MINIMUM qualifier string */
1093    1       MAX_QUAL_S      CHAR(7) INIT('MAXIMUM'),                        /* MAXIMUM qualifier string */
1094    1       TOPC_QUAL_S     CHAR(6) INIT('TOPCPU'),                         /* TOPCPU qualifier string */
1095    1       TOPD_QUAL_S     CHAR(6) INIT('TOPDIO'),                         /* TOPDIO qualifier string */
1096    1       TOPB_QUAL_S     CHAR(6) INIT('TOPBIO'),                         /* TOPBIO qualifier string */
1097    1       TOPF_QUAL_S     CHAR(8) INIT('TOPFAULT'),                       /* TOPFAULT qualifier string */
1098    1       CPU_QUAL_S      CHAR(3) INIT('CPU'),                            /* CPU qualifier string */
1099    1       PCENT_QUAL_S    CHAR(7) INIT('PERCENT'),                        /* PERCENT qualifier string */
1100    1       ITEM_QUAL_S     CHAR(4) INIT('ITEM');                           /* ITEM qualifier string */
1101    1
1102    1
1103 :  1       /*
1104 :  1       /*      Default file-spec values for qualifiers. Descriptors for these strings are
1105 :  1       /*      defined in the DEF structure.
```

```
1106 :  1       /*/
1107    1
1108    1       DECLARE
1109    1
1110    1       REC_DEF_S       CHAR(11) INIT('MONITOR.DAT'),           /* RECORD qualifier default value */
1111    1       DISP_DEF_S      CHAR(11) INIT('SYS$OUTPUT:'),           /* DISPLAY qualifier default value */
1112    1       SUMM_DEF_S      CHAR(11) INIT('MONITOR.SUM');           /* SUMMARY qualifier default value */
1113    1
1114    1
```

```
1115    1
1116    1       DECLARE
1117    1
1118    1       01 MRB_DEF,                                             /* Default Monitor Request Block */
1119    1
1120    1           02 beg      bit(64) aligned         init('0'B),     /* Beginning time of request in system time units */
1121    1           02 end      bit(64) aligned         init('0'B),     /* Ending time of request in system time units */
1122    1           02 int      fixed binary(31)        init(0),        /* Interval value in seconds */
1123    1           02 flush    fixed binary(31)        init(0),        /* Flush interval in seconds */
1124    1           02 vie      fixed binary(31)        init(0),        /* Viewing time for a screen in seconds */
1125    1           02 inp      pointer                 init(null()),   /* Address of input file descr (0 if input not requested) */
1126    1           02 dis      pointer,                                /* Address of display file descr (0 if display not requested
1127  : 1                                                               /* (Initialized in MONITOR_INIT routine) */
1128    1           02 rec      pointer                 init(null()),   /* Address of record file descr (0 if record not requested)
1129    1           02 sum      pointer                 init(null()),   /* Address of summary file descr (0 if summary not requested
1130    1           02 com      pointer                 init(null()),   /* Address of comment string descriptor */
1131    1           02 clc      fixed binary(15),                       /* Count of classes requested (needs no init) */
1132    1           02 clb      bit(128) aligned        init('0'B),     /* Bit string of requested classes */
1133    1           02 inf      fixed binary(7),                        /* Count of input files specified (needs no init) */
1134    1           02 fla,                                             /* Flags for MRB */
1135    1                03 dis  bit(1)                  init(NO),
1136    1                03 rec  bit(1)                  init(NO),
1137    1                03 sum  bit(1)                  init(NO),
1138    1                03 pbk  bit(1)                  init(NO),
1139    1                03 ide  bit(1)                  init(NO),
1140    1                03 dtf  bit(1)                  init(NO),
1141    1                03 icr  bit(1)                  init(NO),
1142    1                03 rcr  bit(1)                  init(NO),
1143    1                03 dcr  bit(1)                  init(NO),
1144    1                03 scr  bit(1)                  init(NO),
1145    1                03 alc  bit(1)                  init(NO),
1146    1                03 mfs  bit(1)                  init(NO),
1147    1                03 byn  bit(1)                  init(NO),
1148    1                03 scl  bit(1)                  init(NO),
1149    1                03 fil  bit(2)                  init('0'B);
1150    1
```

```
1151  1 1     /*
1152  1 1     /*++
1153  1 1     /*
1154  1 1     /* FUNCTIONAL DESCRIPTION:
1155  1 1     /*
1156  1 1     /*      MONMAIN
1157  1 1     /*
1158  1 1     /*      This routine is the main Monitor routine, entered from DCL.
1159  1 1     /*      It calls the CLE (Command Language Editor) to parse the
1160  1 1     /*      MONITOR command line. and creates a Monitor Request Block (MRB)
1161  1 1     /*      describing the request. The EXECUTE_REQUEST routine is then
1162  1 1     /*      called to execute the request.
1163  1 1     /*
1164  1 1     /* INPUTS:
1165  1 1     /*
1166  1 1     /*      None
1167  1 1     /*
1168  1 1     /* IMPLICIT INPUTS:
1169  1 1     /*
1170  1 1     /*      The MONITOR command line.
1171  1 1     /*
1172  1 1     /* OUTPUTS:
1173  1 1     /*
1174  1 1     /*      None
1175  1 1     /*
1176  1 1     /* ROUTINE VALUE:
1177  1 1     /*
1178  1 1     /*      SS$_NORMAL, or failing MONITOR status.
1179  1 1     /*
1180  1 1     /*--
1181  1 1     /*/
1182  1
```

```
1183    1       ON FINISH;                                              /* On finish, do nothing */
1184    1       ON ANYCONDITION                                         /* On any condition signaled, */
1185    1           BEGIN;
1186    2           DECLARE MON_CODE    FIXED BINARY(31),               /* Monitor message code */
1187    2                   TEMP        FIXED BINARY(31),               /* Temporary scratch area */
1188    2                   ON_FILE     CHAR(100) VARYING,              /* Holds possible file name string */
1189    2                   MNR$_FACNO  FIXED BINARY(31) GLOBALREF VALUE, /* MONITOR facility number */
1190    2                   SIGNALED_ERR ENTRY (ANY VALUE, ANY VALUE, ANY VALUE, ANY); /* Rtn to set up PUTMSGVEC */
1191    2
1192    2           CHF$ARGPTR = ONARGSLIST();                          /* Get signal array pointer */
1193    2           STS$VALUE = CHF$SIG_NAME;                           /* Get code for signaled condition */
1194    2           UNSPEC(TEMP) = STS$FAC_NO;                          /* Convert facility no. to binary in TEMP */
1195    2           IF TEMP = MNR$_FACNO                                /* If a MONITOR code, re-signal it */
1196    2               THEN CALL RESIGNAL();
1197    2
1198    2           IF CURR_ERRCODE = MNR$_ERRPARSE &                   /* If expecting a CLU parsing error, */
1199    2              CHF$SIG_NAME = CLI$_NOCOMD                       /* AND it's a "No Command on Line", */
1200    2               THEN CALL = NORMAL;                             /*   then set up code so it's ignored */
1201    2               ELSE DO;                                        /* Otherwise */
1202    3                    IF CURR_ERRCODE = MNR$_ERREXEREA           /* Was it an execute command file read error? */
1203    3                        THEN DO;
1204    4                            CLOSE FILE (COMMAND_FILE);         /*  Yes, close the execute command file */
1205    4                            EXECUTE = NO;                      /*  Don't do any more input from the file */
1206    4                            END;
1207    3                    IF CURR_ERRCODE = 0                        /* See if an error is currently expected */
1208    3                        THEN MON_CODE = MNR$_UNEXPERR;         /* No, set "unexpected" code */
1209    3                        ELSE MON_CODE = CURR_ERRCODE;          /* Yes, set currently expected code */
1210    3                    CALL SIGNALED_ERR(MON_CODE,STS$VALUE,DIM(CHF$SIG_ARG,1),CHF$SIG_ARG); /* Log the error */
1211    3                    CALL = MON_CODE;                           /* Set up code for MONITOR request termination */
1212    3                    END;
1213    2
1214    2           CURR_ERRCODE = 0;                                   /* Reset to default MONITOR error code ("unexpected") */
1215    2           GO TO MON_REQ_TERM;                                 /* ... and go terminate (PL/I does an UNWIND) */
1216    2           END;
1217    1
```

```
1218    1       CALL = MONITOR_INIT();                          /* Do image-wide initialization */
1219    1       IF STATUS                                       /* Continue if status OK */
1220    1           THEN CALL = MONITOR_CMD();                  /* Analyze and execute first (DCL-level) MONITOR cmd */
1221    1
1222    1       MON_REQ_TERM:                                   /* MONITOR request termination */
1223    1
1224    1       /*
1225    1       /*      We get to this point by one of three routes:
1226    1       /*
1227    1       /*              1) A MONITOR request has just terminated successfully or with an error status code; or
1228    1       /*              2) A MONITOR request has just terminated with an error that was signaled; or
1229    1       /*              3) The MONITOR_INIT call above terminated with an error status.
1230    1       /*
1231    1       /*      In all three cases we want to do the same thing. That is, to loop prompting for more subcommands
1232    1       /*      as long as the PROMPT indicator is still set to YES. It can be set to NO by an EXIT subcommand,
1233    1       /*      or as a result of the user's striking CTRL/Z (either in response to the MONITOR> prompt, or while
1234    1       /*      a MONITOR request is running). In case 3 above, it will always be set to NO. For all cases, the
1235    1       /*      variable CALL contains the status code of interest and, if an error, the PUTMSG vector (PUTMSGVEC)
1236    1       /*      has been set up with error message information. STATUS is a synonym for the low-order bit of CALL.
1237    1       /*
1238    1       /*      If the EXECUTE indicator is set to YES then NEXT_EXECUTE_COMMAND will be called.  If EXECUTE is set
1239    1       /*      to NO, then NEXT_COMMAND is called.  NEXT_EXECUTE_COMMAND will retrieve commands from a file intead
1240    1       /*      of the terminal as done by EXECUTE_COMMAND.
1241    1       /*/
1242    1
1243    1       IF STATUS = NOT_SUCCESSFUL                      /* If bad status, */
1244    1           THEN DO;
1245    2                IF DISPLAYING = YES                    /* If display output is active, */
1246    2                    THEN STS$VALUE = DISPLAY_CLEANUP(); /*   then perform cleanup */
1247    2                CALL SIGNAL_MON_ERR();                 /* Signal MONITOR error */
1248    2                END;
1249    1
1250    1       DO WHILE (PROMPT = YES);                        /* Main loop to perform subcommands */
1251    2           IF EXECUTE = YES                            /* Read from the execute command file? */
1252    2               THEN CALL = NEXT_EXECUTE_COMMAND();     /* Yes, execute next subcommand line from the file */
1253    2               ELSE CALL = NEXT_COMMAND();             /* No, Read from the terminal and execute next subcommand li
1254    2           IF STATUS = NOT_SUCCESSFUL THEN DO;         /* If bad status, */
1255    3               IF DISPLAYING = YES                     /* If display output is active, */
1256    3                   THEN STS$VALUE = DISPLAY_CLEANUP(); /*   then perform cleanup */
1257    3               CALL SIGNAL_MON_ERR();                  /* Signal MONITOR error, using PUTMSGVEC */
1258    3               END;
1259    2       END;                                            /* End of subcommand loop */
1260    1
1261    1
1262    1       STS$VALUE = CALL;                               /* Get MONITOR completion status */
1263    1       STS$INHIB_MSG = YES;                            /* Inhibit DCL print */
1264    1       RETURN(STS$VALUE);                              /* Return to DCL */
1265    1
```

```
1266    1       MONITOR_INIT: Procedure Returns(Fixed Binary(31));
1267    2
1268    2       /*
1269    2       /*      This routine performs general set-up, including
1270    2       /*      setting of the current MRB to default values.
1271    2       /*/
1272    2
1273    2
```

```
1274    2
1275    2       PROMPT = NO;                                    /* Start out assuming no MONITOR> prompting */
1276    2       EXECUTE = NO;                                   /* Start out assuming no EXECUTE file */
1277    2       FIRST_MON_CMD = YES;                            /* Indicate first MONITOR command (entered at DCL) */
1278    2       NORMAL = SS$_NORMAL;                            /* Use system's normal status */
1279    2
1280    2       /*
1281    2       /*      Allocate the two blocks which contain string descriptors for
1282    2       /*      command qualifiers and default qualifier values. Global
1283    2       /*      pointers to the blocks are automatically established.
1284    2       /*/
1285    2
1286    2       ALLOCATE QUALIFIER_DESC;
1287    2       ALLOCATE DEF_DESC;
1288    2
1289    2
1290    2       /*
1291    2       /*      Initialize string descriptors for command
1292    2       /*      qualifiers, command parameters, etc.
1293    2       /*/
1294    2
1295    2       QUAL$L_BEG = LENGTH(BEG_QUAL_S);                /* Init length longword of descr */
1296    2       QUAL$A_BEG = ADDR(BEG_QUAL_S);                  /* Init address longword of descr */
1297    2
1298    2       QUAL$L_END = LENGTH(END_QUAL_S);                /* Init length longword of descr */
1299    2       QUAL$A_END = ADDR(END_QUAL_S);                  /* Init address longword of descr */
1300    2
1301    2       QUAL$L_INT = LENGTH(INT_QUAL_S);                /* Init length longword of descr */
1302    2       QUAL$A_INT = ADDR(INT_QUAL_S);                  /* Init address longword of descr */
1303    2
1304    2       QUAL$L_FLUSH = LENGTH(FLUSH_QUAL_S);            /* Init length longword of descr */
1305    2       QUAL$A_FLUSH = ADDR(FLUSH_QUAL_S);              /* Init address longword of descr */
1306    2
1307    2       QUAL$L_VIEW = LENGTH(VIEW_QUAL_S);              /* Init length longword of descr */
1308    2       QUAL$A_VIEW = ADDR(VIEW_QUAL_S);                /* Init address longword of descr */
1309    2
1310    2       QUAL$L_BY_NODE = LENGTH(BY_NODE_QUAL_S);        /* Init length longword of descr */
1311    2       QUAL$A_BY_NODE = ADDR(BY_NODE_QUAL_S);          /* Init address longword of descr */
1312    2
1313    2       QUAL$L_INP = LENGTH(INP_QUAL_S);                /* Init length longword of descr */
1314    2       QUAL$A_INP = ADDR(INP_QUAL_S);                  /* Init address longword of descr */
1315    2
```

```
1316    2       QUAL$L_DISP = LENGTH(DISP_QUAL_S);          /* Init length longword of descr */
1317    2       QUAL$A_DISP = ADDR(DISP_QUAL_S);            /* Init address longword of descr */
1318    2
1319    2       QUAL$L_REC = LENGTH(REC_QUAL_S);            /* Init length longword of descr */
1320    2       QUAL$A_REC = ADDR(REC_QUAL_S);              /* Init address longword of descr */
1321    2
1322    2       QUAL$L_SUMM = LENGTH(SUMM_QUAL_S);          /* Init length longword of descr */
1323    2       QUAL$A_SUMM = ADDR(SUMM_QUAL_S);            /* Init address longword of descr */
1324    2
1325    2       QUAL$L_COMM = LENGTH(COMM_QUAL_S);          /* Init length longword of descr */
1326    2       QUAL$A_COMM = ADDR(COMM_QUAL_S);            /* Init address longword of descr */
1327    2
1328    2       QUAL$L_BY_NODE = LENGTH(BY_NODE_QUAL_S);    /* Init length longword of descr */
1329    2       QUAL$A_BY_NODE = ADDR(BY_NODE_QUAL_S);      /* Init address longword of descr */
1330    2
1331    2       QUAL$L_CLASS = LENGTH(CLASS_PARM_S);        /* Init length longword of descr */
1332    2       QUAL$A_CLASS = ADDR(CLASS_PARM_S);          /* Init address longword of descr */
1333    2
1334    2       QUAL$L_ALL = LENGTH(ALL_QUAL_S);            /* Init length longword of descr */
1335    2       QUAL$A_ALL = ADDR(ALL_QUAL_S);              /* Init address longword of descr */
1336    2
1337    2       QUAL$L_CUR = LENGTH(CUR_QUAL_S);            /* Init length longword of descr */
1338    2       QUAL$A_CUR = ADDR(CUR_QUAL_S);              /* Init address longword of descr */
1339    2
1340    2       QUAL$L_AVE = LENGTH(AVE_QUAL_S);            /* Init length longword of descr */
1341    2       QUAL$A_AVE = ADDR(AVE_QUAL_S);              /* Init address longword of descr */
1342    2
1343    2       QUAL$L_MIN = LENGTH(MIN_QUAL_S);            /* Init length longword of descr */
1344    2       QUAL$A_MIN = ADDR(MIN_QUAL_S);              /* Init address longword of descr */
1345    2
1346    2       QUAL$L_MAX = LENGTH(MAX_QUAL_S);            /* Init length longword of descr */
1347    2       QUAL$A_MAX = ADDR(MAX_QUAL_S);              /* Init address longword of descr */
1348    2
1349    2       QUAL$L_TOPC = LENGTH(TOPC_QUAL_S);          /* Init length longword of descr */
1350    2       QUAL$A_TOPC = ADDR(TOPC_QUAL_S);            /* Init address longword of descr */
1351    2
1352    2       QUAL$L_TOPD = LENGTH(TOPD_QUAL_S);          /* Init length longword of descr */
1353    2       QUAL$A_TOPD = ADDR(TOPD_QUAL_S);            /* Init address longword of descr */
1354    2
1355    2       QUAL$L_TOPB = LENGTH(TOPB_QUAL_S);          /* Init length longword of descr */
1356    2       QUAL$A_TOPB = ADDR(TOPB_QUAL_S);            /* Init address longword of descr */
1357    2
1358    2       QUAL$L_TOPF = LENGTH(TOPF_QUAL_S);          /* Init length longword of descr */
1359    2       QUAL$A_TOPF = ADDR(TOPF_QUAL_S);            /* Init address longword of descr */
1360    2
1361    2       QUAL$L_CPU = LENGTH(CPU_QUAL_S);            /* Init length longword of descr */
1362    2       QUAL$A_CPU = ADDR(CPU_QUAL_S);              /* Init address longword of descr */
1363    2
1364    2       QUAL$L_PCENT = LENGTH(PCENT_QUAL_S);        /* Init length longword of descr */
1365    2       QUAL$A_PCENT = ADDR(PCENT_QUAL_S);          /* Init address longword of descr */
1366    2
1367    2       QUAL$L_ITEM = LENGTH(ITEM_QUAL_S);          /* Init length longword of descr */
1368    2       QUAL$A_ITEM = ADDR(ITEM_QUAL_S);            /* Init address longword of descr */
1369    2
1370    2       DEF$L_REC = LENGTH(REC_DEF_S);              /* Init length longword of descr */
1371    2       DEF$A_REC = ADDR(REC_DEF_S);                /* Init address longword of descr */
```

```
1372     2
1373     2        DEF$L_DISP = LENGTH(DISP_DEF_S);          /* Init length longword of descr */
1374     2        DEF$A_DISP = ADDR(DISP_DEF_S);            /* Init address longword of descr */
1375     2
1376     2        DEF$L_SUMM = LENGTH(SUMM_DEF_S);          /* Init length longword of descr */
1377     2        DEF$A_SUMM = ADDR(SUMM_DEF_S);            /* Init address longword of descr */
1378     2
1379     2
```

```
1380  1  2      /*
1381  1  2      /*      Initialize STAT_TABLE descriptor pointers.
1382  1  2      /*/
1383     2
1384     2      STAT_DESC(ALL_STAT) = ADDR(QUAL$L_ALL);          /* Init pointer to ALL descr */
1385     2      STAT_DESC(CUR_STAT) = ADDR(QUAL$L_CUR);          /* Init pointer to CUR descr */
1386     2      STAT_DESC(AVE_STAT) = ADDR(QUAL$L_AVE);          /* Init pointer to AVE descr */
1387     2      STAT_DESC(MIN_STAT) = ADDR(QUAL$L_MIN);          /* Init pointer to MIN descr */
1388     2      STAT_DESC(MAX_STAT) = ADDR(QUAL$L_MAX);          /* Init pointer to MAX descr */
1389     2
1390  1  2      /*
1391  1  2      /*      Initialize PROCD_TABLE descriptor pointers.
1392  1  2      /*/
1393     2
1394     2      PROCD_DESC(REG_PROC) = NULL();                   /* Indicate no qualifier for "regular" PROCESSES display */
1395     2      PROCD_DESC(TOPC_PROC) = ADDR(QUAL$L_TOPC);       /* Init pointer to TOPC descr */
1396     2      PROCD_DESC(TOPD_PROC) = ADDR(QUAL$L_TOPD);       /* Init pointer to TOPD descr */
1397     2      PROCD_DESC(TOPB_PROC) = ADDR(QUAL$L_TOPB);       /* Init pointer to TOPB descr */
1398     2      PROCD_DESC(TOPF_PROC) = ADDR(QUAL$L_TOPF);       /* Init pointer to TOPF descr */
1399     2
1400     2      MRB_DEF.DIS = ADDR(DEF$L_DISP);                  /* Init default value of ptr to display file descr */
1401     2      ALLOCATE MCA SET(MCAPTR);                        /* Obtain Monitor Communication Area */
1402     2      ALLOCATE SYS_INFO SET(SPTR);                     /* Obtain System Information Area */
1403     2
1404  1  2      /*
1405  1  2      /*      Allocate "temp", "current" and "active" MRBs. Another MRB (MRB_DEF),
1406  1  2      /*      the "default" MRB, is allocated in the data section above. Each MRB fully describes
1407  1  2      /*      a MONITOR request. The usage of the MRBs is analogous to that of SYSGEN PAR files.
1408  1  2      /*      To wit, the "default" MRB never changes; it is moved to the "current" MRB whenever
1409  1  2      /*      an INITIALIZE subcommand is issued; an implicit INITIALIZE occurs when the MONITOR
1410  1  2      /*      image is invoked. The "current" MRB may be changed repeatedly with SET commands without
1411  1  2      /*      actually executing a request; When a MONITOR request is executed (with a MONITOR
1412  1  2      /*      subcommand, or a MONITOR DCL command), the "current" MRB is moved to the "active" MRB,
1413  1  2      /*      the "active" MRB is updated in accordance with the MONITOR request, and the request
1414  1  2      /*      is executed using the "active" MRB. The "temp" MRB is used to accumulate information
1415  1  2      /*      resulting from a SET command. If the command contains no errors, "temp" is swapped
1416  1  2      /*      with "current"; otherwise, "temp" is discarded. This is done to avoid defining a partial
1417  1  2      /*      request when the SET command contains an error.
1418  1  2      /*/
1419     2
1420     2      ALLOCATE MRB SET(TEMP_MRBPTR);                   /* Allocate the "temp" MRB */
1421     2      ALLOCATE MRB SET(CURR_MRBPTR);                   /* Allocate the "current" MRB */
1422     2      ALLOCATE MRB SET(ACT_MRBPTR);                    /* Allocate the "active" MRB */
1423     2      DEF_MRBPTR = ADDR(MRB_DEF);                      /* Init pointer to "default" MRB */
1424     2
1425     2      CALL = INIT_CMD();                               /* Do an INITIALIZE command ... i.e., move default */
1426  1  2                                                       /* ... MRB to current MRB */
1427     2
1428     2      RETURN(CALL);                                    /* Return with status from INIT_CMD */
1429     2      END MONITOR_INIT;
1430     1
```

```
1431    1       NEXT_COMMAND: Procedure Returns(Fixed Binary(31));      /* Routine to read the next subcommand & execute it */
1432    2
1433    2       /***
1434    2       /*
1435    2       /* FUNCTIONAL DESCRIPTION:
1436    2       /*
1437    2       /*      NEXT_COMMAND
1438    2       /*
1439    2       /*      This routine is called by the main routine to read the next subcommand and execute it.
1440    2       /*      It returns a status code indicating the disposition of the command.
1441    2       /*
1442    2       /* INPUTS:
1443    2       /*
1444    2       /*      None
1445    2       /*
1446    2       /* IMPLICIT INPUTS:
1447    2       /*
1448    2       /*      PROMPT -- a bit indicating whether or not to prompt for a command.
1449    2       /*                It has always been set to YES before entry to NEXT_COMMAND.
1450    2       /*
1451    2       /* OUTPUTS:
1452    2       /*
1453    2       /*      None
1454    2       /*
1455    2       /* IMPLICIT OUTPUTS:
1456    2       /*
1457    2       /*      The next command is executed.
1458    2       /*
1459    2       /*      PROMPT  -- a bit indicating whether or not to prompt for another
1460    2       /*      command. It is always set to YES before entry to NEXT_COMMAND, and
1461    2       /*      can be set to NO by NEXT_COMMAND in any of the following situations:
1462    2       /*
1463    2       /*              1) The user issues the EXIT subcommand.
1464    2       /*              2) The user strikes CTRL/Z in response to the MONITOR> prompt.
1465    2       /*              3) The user strikes CTRL/Z while a MONITOR subcommand is running.
1466    2       /*
1467    2       /*      PUTMSGVEC -- a 20-longword vector which is loaded with error message
1468    2       /*                   information by NEXT_COMMAND whenever an error status is
1469    2       /*                   returned to the caller of NEXT_COMMAND.
1470    2       /*
1471    2       /* ROUTINE VALUE:
1472    2       /*
1473    2       /*      A status code indicating the disposition of the command. If an
1474    2       /*      error status, then the PUTMSGVEC error message vector will have
1475    2       /*      been set up.
1476    2       /*
1477    2       /* SIDE EFFECTS:
1478    2       /*
1479    2       /*      None
1480    2       /*
1481    2       /*/
1482    2
```

```
1483  2     /*
1484  2     /*      +-----------------------------------------------------------------------+
1485  2     /*      |                                                                       |
1486  2     /*      |                        COMPILE TIME DEFINITIONS                       |
1487  2     /*      |                                                                       |
1488  2     /*      +-----------------------------------------------------------------------+
1489  2     /*/
1490  2
1491  2     %REPLACE        MAX_EXEC_LINE           BY 256;                 /* Max execute file line size */
1492  2
1493  2
1494  2     /*
1495  2     /*      +-----------------------------------------------------------------------+
1496  2     /*      |                                                                       |
1497  2     /*      |                             LOCAL STORAGE                             |
1498  2     /*      |                                                                       |
1499  2     /*      +-----------------------------------------------------------------------+
1500  2     /*/
1501  2
1502  2     Declare
1503  2       CLI$DCL_PARSE EXTERNAL ENTRY(ANY, ANY)                        /* Rtn to do DCL-like parsing of cmd line */
1504  2                     RETURNS(FIXED BINARY(31)),
1505  2       CLI$DISPATCH  EXTERNAL ENTRY                                  /* Rtn to dispatch to latest parsed cmd */
1506  2                     RETURNS(FIXED BINARY(31)),
1507  2       LIB$GET_INPUT EXTERNAL ENTRY(ANY, ANY, FIXED BINARY(15))      /* Rtn to read a line from SYS$INPUT */
1508  2                     OPTIONS(VARIABLE)
1509  2                     RETURNS(FIXED BINARY(31)),
1510  2       STR$POSITION  EXTERNAL ENTRY(ANY, ANY, ANY)                   /* Rtn to locate 1st occurance of a substring */
1511  2                     OPTIONS(VARIABLE)
1512  2                     RETURNS(FIXED BINARY(31)),
1513  2       STR$REPLACE   EXTERNAL ENTRY(ANY, ANY, ANY, ANY, ANY)         /* Rtn to replace a substring with another substring
1514  2                     RETURNS(FIXED BINARY(31));
1515  2
1516  2     Declare
1517  2     MNR$_ERREXEREP  FIXED BINARY(31) GLOBALREF VALUE,               /* Error message code */
1518  2     MNR$_ERRPROMPT  FIXED BINARY(31) GLOBALREF VALUE;               /* Error message code */
1519  2
1520  2     /*
1521  2     /*      String descriptor for user prompt string
1522  2     /*/
1523  2
1524  2     Declare
1525  2
1526  2     1 PROMPT_STR GLOBALREF,                                         /* User prompt string descriptor */
1527  2       2 L            FIXED BINARY(31),
1528  2       2 A            POINTER;
1529  2
1530  2     Declare
1531  2     1 DYN_STRING    GLOBALREF,                                      /* Dynamic string descriptor */
1532  2       2 L            FIXED BINARY(15),                              /* Length */
1533  2       2 TC           CHAR(2),                                       /* Type and Class */
1534  2       2 A            POINTER,                                       /* Address */
1535  2     DYN_STRING_S    CHAR(DYN_STRING.L) BASED(DYN_STRING.A);         /* String */
1536  2
1537  2     Declare
1538  2     1 CMD_LINE,                                                     /* Static command line descriptor (for subcommands)
```

```
1539    2         2 L             FIXED BINARY(31),                       /* Length */
1540    2         2 A             POINTER,                                /* Address */
1541    2
1542    2       1 REP_LINE,                                               /* Static command line descriptor (for subcommands)
1543    2         2 L             FIXED BINARY(31),                       /* Length */
1544    2         2 A             POINTER,                                /* Address */
1545    2
1546    2       CMD_LINE_S        CHAR(MAX_EXEC_LINE) STATIC;             /* Command buffer to replace "@" with "EXECUTE " */
1547    2
1548    2
1549    2       Declare
1550    2         MONSUB          CHAR(1) GLOBALREF;                      /* Command language definition tables ... */
1551    2                                                                 /* Note -- we simply need a global reference to the
1552    2                                                                 /* tables. Their length is unknown and irrelevant */
1553    2       Declare
1554    2         AT_SIGN_S       CHAR(1) STATIC INIT('@'),               /* "@" used to search command line */
1555    2         EXECUTE_S       CHAR(8) STATIC INIT('EXECUTE '),        /* "EXECUTE " used to replace the "@" in the command
1556    2         AT_SIGN_POS     FIXED BINARY(31);                       /* Position of "@" in the command line */
1557    2
1558    2
1559    2       FIRST_MON_CMD = NO;                                       /* First MONITOR cmd executes before NEXT_COMMAND rt
1560    2       DYN_STRING.L = 0;                                         /* Init cmd line length to enter loop */
1561    2
1562    2       DO WHILE (DYN_STRING.L = 0);                              /* Loop while user enters null lines */
1563    3       CALL = LIB$GET_INPUT(DYN_STRING,PROMPT_STR,);             /* Read the next subcommand */
1564    3       IF STATUS = NOT_SUCCESSFUL                                /* If LIB$GET_INPUT call failed, */
1565    3           THEN DO;
1566    4                PROMPT = NO;                                     /* Indicate no more prompting */
1567    4                IF CALL = RMS$_EOF                               /* If end-of-input, */
1568    4                    THEN RETURN(NORMAL);                         /*   then return with normal status */
1569    4                    ELSE DO;
1570    5                         CALL MON_ERR(MNR$_ERRPROMPT,CALL);      /* Otherwise, log the error ... */
1571    5                         RETURN(MNR$_ERRPROMPT);                 /* and return with status */
1572    5                         END;
1573    4                END;
1574    3       END;
1575    2
1576    2       CMD_LINE.L = DYN_STRING.L;                                /* Copy the length of the command line */
1577    2       CMD_LINE.A = ADDR(CMD_LINE_S);                            /* Get the address of the new working buffer */
1578    2       CMD_LINE_S = DYN_STRING_S;                                /* Copy the command line into the buffer */
1579    2       AT_SIGN_POS = STR$POSITION(CMD_LINE, DESCRIPTOR(AT_SIGN_S));  /* Locate a "@" in the command line */
1580    2       IF AT_SIGN_POS > 0                                        /* Was there one? */
1581    2           THEN DO;                                              /* Yes, prepare to replace the "@" with "EXECUTE " *
1582    3                REP_LINE.L = CMD_LINE.L + 7;                     /* Add space for "EXECUTE " in  replacement desc */
1583    3                REP_LINE.A = CMD_LINE.A;                         /* Get address of replacement string */
1584    3                CALL = STR$REPLACE (REP_LINE, CMD_LINE, AT_SIGN_POS,  /* Replace the "@" with "EXECITE " */
1585    3                                    AT_SIGN_POS, DESCRIPTOR(EXECUTE_S));/*  (need REP_LINE to prevent trunc warning) */
1586    3                IF STATUS = NOT_SUCCESSFUL                       /* If STR$REPLACE call failed, */
1587    3                    THEN DO;
1588    4                         CALL MON_ERR(MNR$_ERREXEREP,CALL);      /* Log the error ... */
1589    4                         RETURN(MNR$_ERREXEREP);                 /* and return with status */
1590    4                         END;
1591    3                    ELSE CMD_LINE.L = REP_LINE.L;                /* STR$REPLACE succeeded, update length of descr */
1592    3                END;
1593    2
```

```
1594    2       CURR_ERRCODE = MNR$_ERRPARSE;                   /* Set MONITOR code in case parsing error signaled *
1595    2       CALL = CLI$DCL_PARSE(CMD_LINE,MONSUB);          /* Parse the subcommand */
1596    2       CURR_ERRCODE = 0;                               /* Reset to default MONITOR code */
1597    2       IF STATUS = NOT_SUCCESSFUL                      /* If parse failed, */
1598    2           THEN DO;
1599    3                IF CALL = CLI$_NOCOMD                  /* If "no command on line" */
1600    3                    THEN RETURN(NORMAL);               /*   then quietly ignore it */
1601    3                    ELSE DO;                           /* Otherwise, */
1602    4                         CALL MON_ERR(MNR$_ERRPARSE,CALL);  /*   log the error ... */
1603    4                         RETURN(MNR$_ERRPARSE);        /*   ... and return with status */
1604    4                         END;
1605    3                END;
1606    2
1607    2       CALL = CLI$DISPATCH();                          /* Execute the parsed command */
1608    2                                                       /* Note -- command subroutines return status */
1609    2                                                       /*         values and log their own errors by */
1610    2                                                       /*         calling MON_ERR */
1611    2
1612    2       CURR_ERRCODE = 0;                               /* Reset to default MONITOR code in case subcommand
1613    2       RETURN(CALL);                                   /* Return to caller with cmd subroutine's status */
1614    2
1615    2       END NEXT_COMMAND;
1616    1
1617    1
```

```
1618    1       NEXT_EXECUTE_COMMAND: Procedure Returns(Fixed Binary(31));      /* Routine to read the next subcommand & execute it
1619    2
1620    2       /***
1621    2       /*
1622    2       /* FUNCTIONAL DESCRIPTION:
1623    2       /*
1624    2       /*      NEXT_EXECUTE_COMMAND
1625    2       /*
1626    2       /*      This routine is called by the main routine to read the next subcommand from
1627    2       /*      the execute command file and execute it.  It returns a status code indicating
1628    2       /*      the disposition of the command.
1629    2       /*
1630    2       /* INPUTS:
1631    2       /*
1632    2       /*      None
1633    2       /*
1634    2       /* IMPLICIT INPUTS:
1635    2       /*
1636    2       /*      COMMAND_FILE - file reference used to read the execute command file.
1637    2       /*
1638    2       /*      EXECUTE - a bit indicating whether or not commands are read from an execute
1639    2       /*                command file.  It has been set to YES before entry to NEXT_EXECUTE_COMMAND.
1640    2       /*
1641    2       /* OUTPUTS:
1642    2       /*
1643    2       /*      None
1644    2       /*
1645    2       /* IMPLICIT OUTPUTS:
1646    2       /*
1647    2       /*      The next command is executed.
1648    2       /*
1649    2       /*      EXECUTE - a bit indicating whether or not to read from the execute command file
1650    2       /*      for another command.  It is always set to YES before entry to NEXT_EXECUTE_COMMAND,
1651    2       /*      and can be set to NO by NEXT_EXECUTE_COMMAND in any of the following situations:
1652    2       /*
1653    2       /*              1) Execute command file EOF is encountered.
1654    2       /*              2) The user issues the EXIT subcommand.
1655    2       /*              3) The user strikes CTRL/Z in response to the MONITOR> prompt.
1656    2       /*              4) The user strikes CTRL/Z while a MONITOR subcommand is running.
1657    2       /*              5) The file cannot be opened properly.
1658    2       /*              6) An error occurs when trying to read the file.
1659    2       /*
1660    2       /*      PUTMSGVEC -- a 20-longword vector which is loaded with error message
1661    2       /*                   information by NEXT_EXECUTE_COMMAND whenever an error status is
1662    2       /*                   returned to the caller of NEXT_EXECUTE_COMMAND.
1663    2       /*
1664    2       /* ROUTINE VALUE:
1665    2       /*
1666    2       /*      A status code indicating the disposition of the command. If an error status,
1667    2       /*      then the PUTMSGVEC error message vector will have been set up.
1668    2       /*
1669    2       /*      MNR$_ERRPARSE   - error parsing command from the execute command file.
1670    2       /*      MNR$_ERREXEREA  - error reading execute command file.
1671    2       /*      SS$_NORMAL      - Success.
1672    2       /*
1673    2       /* SIDE EFFECTS:
```

```
1674  :  2      /*
1675  :  2      /*      None.
1676  :  2      /*
1677  :  2      /*/
1678     2
```

```
1679  2      /*
1680  2      /*      +-----------------------------------------------------------------------+
1681  2      /*      |                                                                       |
1682  2      /*      |                        COMPILE TIME DEFINITIONS                       |
1683  2      /*      |                                                                       |
1684  2      /*      +-----------------------------------------------------------------------+
1685  2      /*/
1686  2
1687  2      %REPLACE        MAX_EXEC_LINE           BY 256;                 /* Max execute file line size */
1688  2
1689  2      /*
1690  2      /*      +-----------------------------------------------------------------------+
1691  2      /*      |                                                                       |
1692  2      /*      |                        EXTERNAL STORAGE DEFINITIONS                   |
1693  2      /*      |                                                                       |
1694  2      /*      +-----------------------------------------------------------------------+
1695  2      /*/
1696  2
1697  2      Declare
1698  2          CLI$DCL_PARSE       EXTERNAL ENTRY(ANY, ANY)                    /* Rtn to do DCL-like parsing of cmd line */
1699  2                              RETURNS(FIXED BINARY(31)),
1700  2          CLI$DISPATCH        EXTERNAL ENTRY                              /* Rtn to dispatch to latest parsed cmd */
1701  2                              RETURNS(FIXED BINARY(31)),
1702  2          CLI$GET_VALUE       ENTRY(ANY, ANY)                             /* CLE routine to get qualifier values */
1703  2                              RETURNS(FIXED BINARY(31)),
1704  2          STR$POSITION        EXTERNAL ENTRY(ANY, ANY, ANY)               /* Rtn to locate 1st occurance of a substring */
1705  2                              OPTIONS(/ARIABLE)
1706  2                              RETURNS(FIXED BINARY(31)),
1707  2          STR$REPLACE         EXTERNAL ENTRY(ANY, ANY, ANY, ANY, ANY)     /* Rtn to replace a substring with another substring
1708  2                              RETURNS(FIXED BINARY(31));
1709  2
1710  2
1711  2      Declare
1712  2          COMMAND_FILE        FILE                GLOBALREF,          /* Execute Command File */
1713  2          MONSUB              CHAR(1)             GLOBALREF,          /* Command language definition tables ... */
1714  2                                                                      /* Note -- we simply need a global reference to the
1715  2                                                                      /* tables. Their length is unknown and irrelevant */
1716  2          NORMAL              FIXED BINARY(31)    GLOBALREF,          /* MONITOR normal return status */
1717  2          EXECUTE             BIT(1) ALIGNED      GLOBALREF;          /* YES => read another command from the execute file
1718  2
1719  2      Declare
1720  2          MNR$_ERREXEREP      FIXED BINARY(31) GLOBALREF VALUE,       /* Error message code */
1721  2          MNR$_ERREXEREA      FIXED BINARY(31) GLOBALREF VALUE,       /* Error message code */
1722  2          MNR$_ERRPARSE       FIXED BINARY(31) GLOBALREF VALUE;       /* Error message code */
1723  2
1724  2
1725  2      /*
1726  2      /*      +-----------------------------------------------------------------------+
1727  2      /*      |                                                                       |
1728  2      /*      |                        GLOBAL STORAGE DEFINITIONS                     |
1729  2      /*      |                                                                       |
1730  2      /*      +-----------------------------------------------------------------------+
1731  2      /*/
1732  2
1733  2      Declare
1734  2          1 SUB_COMMAND       GLOBALDEF,
```

```
1735    2           2 L                 FIXED BINARY(31),
1736    2           2 A                 POINTER,
1737    2       COMMAND_RECORD          CHAR(MAX_EXEC_LINE) STATIC VARYING GLOBALDEF;   /* Buffer for a subcommand */
1738    2
1739    2
1740    2       /*
1741    2       /*      |-------------------------------------------------------------------------+
1742    2       /*      |                                                                         |
1743    2       /*      |                              LOCAL STORAGE                              |
1744    2       /*      |                                                                         |
1745    2       /*      |-------------------------------------------------------------------------+
1746    2       /*/
1747    2
1748    2
1749    2       Declare
1750    2           1 REP_LINE,                                                 /* Static command line descriptor (for subcommands)
1751    2             2 L                 FIXED BINARY(31),                     /* Length */
1752    2             2 A                 POINTER;                              /* Address */
1753    2
1754    2       Declare
1755    2           AT_SIGN_S   CHAR(1) STATIC INIT('@'),                       /* "@" used to search command line */
1756    2           EXECUTE_S   CHAR(8) STATIC INIT('EXECUTE '),                /* "EXECUTE " used to replace "@" */
1757    2           AT_SIGN_POS FIXED BINARY(31),                               /* Position of "@" in command line */
1758    2           TEMP_COMMAND_PTR    FIXED BINARY(31)                        /* Alias for SUB_COMMAND.A computation */
1759    2                               BASED(ADDR(SUB_COMMAND.A));
1760    2
1761    2
1762    2       ON ENDFILE (COMMAND_FILE) GOTO COMMAND_EOF;                     /* Set up the EOF condition */
1763    2
1764    2
1765    2       CURR_ERRCODE = MNR$_ERREXEREA;                                  /* Set MONITOR code in case read error is signaled *
1766    2       READ FILE (COMMAND_FILE) INTO (COMMAND_RECORD);                 /* Read the next subcommand */
1767    2       CURR_ERRCODE = 0;                                               /* Reset the error code, assume the condition was re
1768    2
1769    2       SUB_COMMAND.L = LENGTH(COMMAND_RECORD);                         /* Set the length of the SUB_COMMAND descriptor */
1770    2       SUB_COMMAND.A = ADDR(COMMAND_RECORD);                           /* Set the address of the SUB_COMMAND descriptor */
1771    2       TEMP_COMMAND_PTR = TEMP_COMMAND_PTR + 2;                        /* Advance ptr beyond length word */
1772    2       AT_SIGN_POS = STR$POSITION(SUB_COMMAND, DESCRIPTOR(AT_SIGN_S)); /* Locate "@" in command line */
1773    2       IF AT_SIGN_POS > 0                                              /* Was there a "@" ? */
1774    2           THEN DO;                                                    /* Yes, prepare to replace "@" with "EXECUTE " */
1775    3                REP_LINE.L = SUB_COMMAND.L + 7;                        /* Add space for "EXECUTE " in replacement string */
1776    3                REP_LINE.A = SUB_COMMAND.A;                            /* Get address of subcommand buffer */
1777    3                CALL = STR$REPLACE (REP_LINE, SUB_COMMAND, AT_SIGN_POS,/* Replace "@" with "EXECUTE " */
1778    3                                    AT_SIGN_POS, DESCRIPTOR(EXECUTE_S));/*  (REP_LINE prevents truncation warning) */
1779    3                IF STATUS = NOT_SUCCESSFUL                             /* If STR$REPLACE call failed, */
1780    3                    THEN DO;
1781    4                        CALL MON_ERR(MNR$_ERREXEREP,CALL);             /* Log the error ... */
1782    4                        RETURN(MNR$_ERREXEREP);                        /* and return with status */
1783    4                        END;
1784    3                    ELSE SUB_COMMAND.L = REP_LINE.L;                   /* STR$REPACE succeded, upade command desc */
1785    3                END;
1786    2
```

```
1787     2      CURR_ERRCODE = MNR$_ERRPARSE;                    /* Set MONITOR code in case parsing error signaled *
1788     2      CALL = CLI$DCL_PARSE(SUB_COMMAND,MONSUB);        /* Parse the subcommand */
1789     2      CURR_ERRCODE = 0;                                /* Reset to default MONITOR code */
1790     2      IF STATUS = NOT_SUCCESSFUL                       /* If parse failed, */
1791     2          THEN DO;
1792     3               IF CALL = CLI$_NOCOMD                   /* If "no command on line" */
1793     3                   THEN RETURN(NORMAL);                /*   then quietly ignore it */
1794     3                   ELSE DO;                            /* Otherwise, */
1795     4                        CALL MON_ERR(MNR$_ERRPARSE,CALL);  /*   log the error ... */
1796     4                        RETURN(MNR$_ERRPARSE);         /*   ... and return with status */
1797     4                        END;
1798     3               END;
1799     2
1800     2      CALL = CLI$DISPATCH();                           /* Execute the parsed command */
1801     2                                                       /* Note -- command subroutines return status */
1802     2                                                       /*         values and log their own errors by */
1803     2                                                       /*         calling MON_ERR */
1804     2      CURR_ERRCODE = 0;                                /* Reset to default MONITOR code in case subcommand
1805     2      RETURN(CALL);                                    /* Return to caller with cmd subroutine's status */
1806     2
1807     2
1808     2      COMMAND_EOF:
1809     2          CLOSE FILE (COMMAND_FILE);                   /* Close the file after EOF condition raised */
1810     2          EXECUTE = NO;                                /* Indicate no more from the execute file */
1811     2          CURR_ERRCODE = 0;                            /* Reset to default MONITOR code in case subcommand
1812     2          RETURN(NORMAL);                              /* Return to caller with cmd subroutine's status */
1813     2
1814     2      END NEXT_EXECUTE_COMMAND;
1815     1
1816     1      END MONMAIN;
1817
```

```
1818          GET_QUALIFIERS: Procedure Returns(Fixed Binary(31));
1819    1
1820    1     /*++
1821    1     /*
1822    1     /* FUNCTIONAL DESCRIPTION:
1823    1     /*
1824    1     /*      GET_QUALIFIERS
1825    1     /*
1826    1     /*      Communicate with CLE to get qualifier settings and
1827    1     /*      their values. Record all such info in the MRB (Monitor
1828    1     /*      Request Block) pointed to by the current value of MRBPTR
1829    1     /*      (it may be the "current" or the "active" MRB).
1830    1     /*
1831    1     /* INPUTS:
1832    1     /*
1833    1     /*      TBS
1834    1     /*
1835    1     /* IMPLICIT INPUTS:
1836    1     /*
1837    1     /*      TBS
1838    1     /*
1839    1     /* OUTPUTS:
1840    1     /*
1841    1     /*      TBS
1842    1     /*
1843    1     /* IMPLICIT OUTPUTS:
1844    1     /*
1845    1     /*      TBS
1846    1     /*
1847    1     /* ROUTINE VALUE:
1848    1     /*
1849    1     /*      TBS
1850    1     /*
1851    1     /* SIDE EFFECTS:
1852    1     /*
1853    1     /*      TBS
1854    1     /*
1855    1     /*/
1856    1
```

```
1857  1     /*
1858  1     /*      ,-------------------------------------------------------------------------.
1859  1     /*      |                                                                         |
1860  1     /*      |                              INCLUDE  FILES                             |
1861  1     /*      |                                                                         |
1862  1     /*      `-------------------------------------------------------------------------'
1863  1     /*/
1864  1
1865  1     %INCLUDE        MONDEF;                         /* Monitor utility structure definitions */
2633  1
2634  1     /*
2635  1     /*      ,-------------------------------------------------------------------------.
2636  1     /*      |                                                                         |
2637  1     /*      |                    SYSTEM SERVICE MACRO DEFINITIONS                     |
2638  1     /*      |                                                                         |
2639  1     /*      `-------------------------------------------------------------------------'
2640  1     /*/
2641  1
2642  1     %INCLUDE        SYS$GETTIM;                     /* $GETTIM system service */
2648  1     /*
2649  1     /*      ,-------------------------------------------------------------------------.
2650  1     /*      |                                                                         |
2651  1     /*      |                      EXTERNAL STORAGE  DEFINITIONS                      |
2652  1     /*      |                                                                         |
2653  1     /*      `-------------------------------------------------------------------------'
2654  1     /*/
2655  1
2656  1     Declare
2657  1     CLI$_PRESENT    FIXED BINARY(31) GLOBALREF VALUE,       /* CLI$PRESENT return status code for "explicitly pr
2658  1     CLI$_NEGATED    FIXED BINARY(31) GLOBALREF VALUE,       /* CLI$PRESENT return status code for "explicitly ne
2659  1     NORMAL          FIXED BINARY(31) GLOBALREF,             /* MONITOR normal return status */
2660  1     MRBPTR          POINTER          GLOBALREF,             /* Pointer to MRB (Monitor Request Block) */
2661  1     M               POINTER          DEFINED(MRBPTR),       /* Short-hand synonym for MRBPTR */
2662  1     DEF_MRBPTR      POINTER          GLOBALREF,             /* Pointer to "default" MRB (Monitor Request Block)
2663  1     MCAPTR          POINTER          GLOBALREF,             /* Pointer to MCA (Monitor Communication Area) */
2664  1     MC              POINTER          DEFINED(MCAPTR),       /* Short-hand synonym for MCAPTR */
2665  1     QUALPTR         POINTER          GLOBALREF,             /* Pointer to Qualifier Descriptors Block */
2666  1     DEFPTR          POINTER          GLOBALREF,             /* Pointer to Qualifier Default Value Descriptors Bl
2667  1     CURR_ERRCODE    FIXED BINARY(31) GLOBALREF;             /* MONITOR error status code currently expected */
2668  1
2669  1     Declare
2670  1     INP_PTR_VOL     POINTER GLOBALREF,                      /* Pointer to volatile /INPUT file-spec */
2671  1     DISP_PTR_VOL    POINTER GLOBALREF,                      /* Pointer to volatile /DISPLAY file-spec */
2672  1     REC_PTR_VOL     POINTER GLOBALREF,                      /* Pointer to volatile /RECORD file-spec */
2673  1     SUMM_PTR_VOL    POINTER GLOBALREF,                      /* Pointer to volatile /SUMMARY file-spec */
2674  1     COMM_PTR_VOL    POINTER GLOBALREF;                      /* Pointer to volatile /COMMENT string */
2675  1
2676  1     Declare
2677  1     INP_PTR_SWAP    BIT(1) ALIGNED GLOBALREF,               /* YES => swap INP_PTR_VOL and INP_PTR_PERM */
2678  1     DISP_PTR_SWAP   BIT(1) ALIGNED GLOBALREF,               /* YES => swap DISP_PTR_VOL and DISP_PTR_PERM */
2679  1     REC_PTR_SWAP    BIT(1) ALIGNED GLOBALREF,               /* YES => swap REC_PTR_VOL and REC_PTR_PERM */
2680  1     SUMM_PTR_SWAP   BIT(1) ALIGNED GLOBALREF,               /* YES => swap SUMM_PTR_VOL and SUMM_PTR_PERM */
2681  1     COMM_PTR_SWAP   BIT(1) ALIGNED GLOBALREF;               /* YES => swap COMM_PTR_VOL and COMM_PTR_PERM */
2682  1
```

```
2683    1       Declare
2684    1       1 DYN_STRING    GLOBALREF,                              /* Dynamic string descriptor */
2685    1         2 L           FIXED BINARY(15),                       /* Length */
2686    1         2 TC          CHAR(2),                                /* Type and Class */
2687    1         2 A           POINTER,                                /* Address */
2688    1
2689    1       DYN_STRING_S    CHAR(DYN_STRING.L) BASED(DYN_STRING.A); /* String */
2690    1
2691    1       /*
2692    1       /*      !-------------------------------------------------------------------------+
2693    1       /*      !                                                                         !
2694    1       /*      !                       EXTERNAL ROUTINE DEFINITIONS                      !
2695    1       /*      !                                                                         !
2696    1       /*      !-------------------------------------------------------------------------+
2697    1       /*/
2698    1
2699    1       Declare
2700    1       CLI$GET_VALUE   ENTRY(ANY, ANY, FIXED BINARY(15))               /* CLE routine to get qualifier values */
2701    1                       OPTIONS(VARIABLE) RETURNS(BIT(1)),
2702    1       CLI$PRESENT     EXTERNAL ENTRY(ANY) RETURNS(FIXED BINARY(31)),  /* CLE routine to determine presence of qualifiers *
2703    1       LIB$CVT_TIME    EXTERNAL ENTRY(ANY, BIT(64) ALIGNED)            /* RTL routine to convert to qword time val */
2704    1                       RETURNS(BIT(1)),
2705    1       STR$UPCASE      EXTERNAL ENTRY(ANY,ANY) RETURNS(BIT(1)),        /* RTL routine to upcase a string */
2706    1       MON_ERR         ENTRY (ANY VALUE, ANY, ANY) OPTIONS(VARIABLE);  /* MONITOR MACRO-32 routine to log synchronous error
2707    1
2708    1       /*
2709    1       /*      !-------------------------------------------------------------------------+
2710    1       /*      !                                                                         !
2711    1       /*      !                         MESSAGE DEFINITIONS                             !
2712    1       /*      !                                                                         !
2713    1       /*      !-------------------------------------------------------------------------+
2714    1       /*/
2715    1
2716    1       Declare
2717    1       MNR$_INVBTIMSP  FIXED BINARY(31) GLOBALREF VALUE,       /* Error message code */
2718    1       MNR$_INVETIMSP  FIXED BINARY(31) GLOBALREF VALUE,       /* Error message code */
2719    1       MNR$_INVINTSP   FIXED BINARY(31) GLOBALREF VALUE,       /* Error message code */
2720    1       MNR$_INVFLUSHSP FIXED BINARY(31) GLOBALREF VALUE,       /* Error message code */
2721    1       MNR$_INVVIEWSP  FIXED BINARY(31) GLOBALREF VALUE,       /* Error message code */
2722    1       MNR$_INVINPFIL  FIXED BINARY(31) GLOBALREF VALUE;       /* Error message code */
2723    1       /*
2724    1       /*      !-------------------------------------------------------------------------+
2725    1       /*      !                                                                         !
2726    1       /*      !                       GLOBAL STORAGE DEFINITIONS                        !
2727    1       /*      !                                                                         !
2728    1       /*      !-------------------------------------------------------------------------+
2729    1       /*/
2730    1
2731    1       Declare
2732    1       QUAL_SPECIFIED  BIT(1) ALIGNED GLOBALDEF INIT('0'B);    /* YES => at least 1 qualifier explicitly spec'd */
2733    1
2734    1       /*
2735    1       /*      !-------------------------------------------------------------------------+
2736    1       /*      !                                                                         !
2737    1       /*      !                         COMPILE-TIME CONSTANTS                          !
2738    1       /*      !                                                                         !
```

```
2739  :  1      /*      !-------------------------------------------------------------------------------+
2740  :  1      /*/
2741     1
2742     1      %REPLACE        SECONDS_TOK_SIZE        BY 7;           /* Size of token for seconds */
2743     1      %REPLACE        TIME_TOK_SIZE           BY 40;          /* Size of token for time specs */
2744     1      %REPLACE        FILE_SPEC_SIZE          BY 128;         /* Max file spec size */
2745     1
```

```
2746  1     /*
2747  1     /*      +-----------------------------------------------------------------------+
2748  1     /*      |                                                                       |
2749  1     /*      |                              OWN STORAGE                              |
2750  1     /*      |                                                                       |
2751  1     /*      +-----------------------------------------------------------------------+
2752  1     /*/
2753  1
2754  1     /*
2755  1     /*      +-----------------------------------------------------------------------+
2756  1     /*      |                                                                       |
2757  1     /*      |                            LOCAL STORAGE                              |
2758  1     /*      |                                                                       |
2759  1     /*      +-----------------------------------------------------------------------+
2760  1     /*/
2761  1
2762  1     Declare
2763  1     TEMP                    FIXED BINARY(31),               /* Temporary "scratch" area */
2764  1     CALL                    FIXED BINARY(31),               /* Holds function value (return status) of called ro
2765  1     STATUS                  BIT(1) BASED(ADDR(CALL));       /* Low-order status bit for called routines */
2766  1
2767  1     Declare
2768  1     FILE_SPEC_PTR           POINTER,                        /* Pointer to structure consisting of file-spec ...
2769  1                                                             /* ... string descriptor followed by string itself *
2770  1     PARSED_SPEC_PTR         POINTER,                        /* Pointer to structure consisting of file-spec ...
2771  1                                                             /* ... string descriptor followed by string itself *
2772  1     COMM_STR_PTR            POINTER;                        /* Pointer to structure consisting of comment ... */
2773  1                                                             /* ... string descriptor followed by string itself *
2774  1
2775  1     Declare
2776  1     1 FILE_SPEC     BASED(FILE_SPEC_PTR),                   /* File-spec string descriptor and string */
2777  1         2 L         FIXED BINARY(31),                       /* File-spec string length */
2778  1         2 A         POINTER,                                /* File-spec string address */
2779  1         2 S         CHAR(FILE_SPEC_SIZE),                   /* File-spec string */
2780  1
2781  1
2782  1     1 PARSED_SPEC   BASED(PARSED_SPEC_PTR),                 /* Parsed File-spec dynamic string descriptor and st
2783  1         2 L         FIXED BINARY(15),                       /* Length */
2784  1         2 T         FIXED BINARY(7),                        /* Type */
2785  1         2 C         FIXED BINARY(7),                        /* Class */
2786  1         2 A         POINTER,                                /* Address */
2787  1
2788  1
2789  1     1 DYN_SPEC      GLOBALDEF,                              /* Dynamic File-spec string descriptor and string */
2790  1         2 L         FIXED BINARY(15),                       /* Length */
2791  1         2 T         FIXED BINARY(7),                        /* Type */
2792  1         2 C         FIXED BINARY(7),                        /* Class */
2793  1         2 A         POINTER;                                /* Address */
2794  1
2795  1
2796  1     Declare
2797  1     1 COMM_STR      BASED(COMM_STR_PTR),                    /* Comment descriptor and string */
2798  1         2 L         FIXED BINARY(31),                       /* Comment string length */
2799  1         2 A         POINTER,                                /* Comment string address */
2800  1         2 S         CHAR(MNR_HDR$K_MAXCOMLEN);              /* Comment string */
2801  1
```

```
2802    1       Declare
2803    1         1 Q_VAL,                                        /* String descriptor for qualifier value */
2804    1           2 L                 FIXED BINARY(31),         /* Length */
2805    1           2 A                 POINTER,                  /* Address */
2806    1
2807    1         Q_VAL_TIME            CHAR(TIME_TOK_SIZE),      /* Qualifier value string for time values */
2808    1         Q_VAL_SECS            CHAR(SECONDS_TOK_SIZE+1), /* Qualifier value string for seconds values */
2809    1         TS_LEN                FIXED BINARY(15);         /* Actual length of time spec */
2810    1
2811    1
```

```
2812    1
2813    1       QUAL_SPECIFIED = NO;                                    /* No qualifiers explicitly specified yet */
2814    1
2815    1       CALL = SYS$GETTIM(MC->MCA$Q_CURR_TIME);                 /* Get current time from system */
2816    1
2817    1       Q_VAL.L = LENGTH(Q_VAL_TIME);                           /* Set length field of descriptor */
2818    1       Q_VAL.A = ADDR(Q_VAL_TIME);                             /* Set address field of descriptor */
2819    1
2820    1       CALL = CLI$PRESENT(QUAL$L_BEG);                         /* Get BEGINNING qualifier presence indicato
2821    1       IF CALL = CLI$_PRESENT                                  /* If explicitly present, */
2822    1         THEN
2823    1           DO;
2824    2           QUAL_SPECIFIED = YES;                               /* Indicate qualifier explicitly specified *
2825    2           IF CLI$GET_VALUE(QUAL$L_BEG,Q_VAL,TS_LEN)           /* Get 'BEGINNING' string and check status *
2826    2           THEN DO;                                            /* Value was specified */
2827    3                Q_VAL.L = TS_LEN;                              /* Pick up actual length */
2828    3                IF Q_VAL.L > TIME_TOK_SIZE                     /* Check for valid size for time spec */
2829    3                    THEN DO;
2830    4                         CALL MON_ERR(MNR$_INVBTIMSP);         /* Log possible error */
2831    4                         RETURN(MNR$_INVBTIMSP);               /* ... and return with status */
2832    4                         END;
2833    3                IF ^ STR$UPCASE(Q_VAL,Q_VAL)                   /* Upcase and check status */
2834    3                    THEN DO;
2835    4                         CALL MON_ERR(MNR$_INVBTIMSP);         /* Log possible error */
2836    4                         RETURN(MNR$_INVBTIMSP);               /* ... and return with status */
2837    4                         END;
2838    3                IF ^ LIB$CVT_TIME(Q_VAL,M->MRB$Q_BEGINNING)    /* Cvt to system time */
2839    3                    THEN DO;
2840    4                         CALL MON_ERR(MNR$_INVBTIMSP);         /* Log possible error */
2841    4                         RETURN(MNR$_INVBTIMSP);               /* ... and return with status */
2842    4                         END;
2843    3                END;
2844    2
2845    2           ELSE                                                /* Value was defaulted */
2846    2                M->MRB$Q_BEGINNING = DEF_MRBPTR->MRB$Q_BEGINNING; /* Store the default value */
2847    2           END;
2848    1
2849    1       Q_VAL.L = LENGTH(Q_VAL_TIME);                           /* Set length field of descriptor */
2850    1
2851    1       CALL = CLI$PRESENT(QUAL$L_END);                         /* Get ENDING qualifier presence indicator *
2852    1       IF CALL = CLI$_PRESENT                                  /* If explicitly present, */
2853    1         THEN
2854    1           DO;
2855    2           QUAL_SPECIFIED = YES;                               /* Indicate qualifier explicitly specified *
2856    2           IF CLI$GET_VALUE(QUAL$L_END,Q_VAL,TS_LEN)           /* Get 'ENDING' string and check status */
2857    2           THEN DO;                                            /* Value was specified */
2858    3                Q_VAL.L = TS_LEN;                              /* Pick up actual length */
2859    3                IF Q_VAL.L > TIME_TOK_SIZE                     /* Check for valid size for time spec */
2860    3                    THEN DO;
2861    4                         CALL MON_ERR(MNR$_INVETIMSP);         /* Log possible error */
2862    4                         RETURN(MNR$_INVETIMSP);               /* ... and return with status */
2863    4                         END;
2864    3                IF ^ STR$UPCASE(Q_VAL,Q_VAL)                   /* Upcase and check status */
2865    3                    THEN DO;
2866    4                         CALL MON_ERR(MNR$_INVETIMSP);         /* Log possible error */
2867    4                         RETURN(MNR$_INVETIMSP);               /* ... and return with status */
```

```
2868    4                   END;
2869    3               IF ^ LIB$CVT_TIME(Q_VAL,M->MRB$Q_ENDING)                /* Cvt to system time */
2870    3                   THEN DO;
2871    4                       CALL MON_ERR(MNR$_INVETIMSP);                   /* Log possible error */
2872    4                       RETURN(MNR$_INVETIMSP);                         /* ... and return with status */
2873    4                       END;
2874    3               END;
2875    2
2876    2           ELSE                                                        /* Value was defaulted */
2877    2               M->MRB$Q_ENDING = DEF_MRBPTR->MRB$Q_ENDING;             /* Store the default value */
2878    2           END;
2879    1
```

```
2880    1       Q_VAL.L = LENGTH(Q_VAL_SECS);                                   /* Set length field of descriptor */
2881    1       Q_VAL.A = ADDR(Q_VAL_SECS);                                     /* Set address field of descriptor */
2882    1
2883    1       CALL = CLI$PRESENT(QUAL$L_INT);                                 /* Get INTERVAL qualifier presence indicator
2884    1       IF CALL = CLI$_PRESENT                                          /* If explicitly present, */
2885    1         THEN
2886    1           DO;
2887    2           QUAL_SPECIFIED = YES;                                       /* Indicate qualifier explicitly specified *
2888    2           IF CLI$GET_VALUE(QUAL$L_INT,Q_VAL)                          /* Get "INTERVAL" string and check status */
2889    2           THEN DO;                                                    /* Value was specified */
2890    3                   Q_VAL.L = INDEX(Q_VAL_SECS,' ') - 1;                /* Eliminate trailing blanks */
2891    3                   IF Q_VAL.L <= 0                                     /* Check for valid size for "seconds" */
2892    3                   THEN DO;
2893    4                        CALL MON_ERR(MNR$_INVINTSP);                   /* Log possible error */
2894    4                        RETURN(MNR$_INVINTSP);                         /* ... and return with status */
2895    4                        END;
2896    3                   CURR_ERRCODE = MNR$_INVINTSP;                       /* Set MONITOR code in case conversion error
2897    3                   M->MRB$L_INTERVAL = BIN(SUBSTR(Q_VAL_SECS,1,Q_VAL.L),31);  /* Convert seconds to binary */
2898    3                   CURR_ERRCODE = 0;                                   /* Reset to default MONITOR code */
2899    3                   IF M->MRB$L_INTERVAL <= 0                           /* Check for valid value */
2900    3                   THEN DO;
2901    4                        CALL MON_ERR(MNR$_INVINTSP);                   /* Log possible error */
2902    4                        RETURN(MNR$_INVINTSP);                         /* ... and return with status */
2903    4                        END;
2904    3                END;
2905    2
2906    2           ELSE                                                        /* Value was defaulted */
2907    2                M->MRB$L_INTERVAL = DEF_MRBPTR->MRB$L_INTERVAL;        /* Store the default value */
2908    2           END;
2909    1
2910    1       Q_VAL.L = LENGTH(Q_VAL_SECS);                                   /* Set length field of descriptor */
2911    1
2912    1       CALL = CLI$PRESENT(QUAL$L_FLUSH);                               /* Get FLUSH qualifier presence indicator */
2913    1       IF CALL = CLI$_PRESENT                                          /* If explicitly present, */
2914    1         THEN
2915    1           DO;
2916    2           QUAL_SPECIFIED = YES;                                       /* Indicate qualifier explicitly specified *
2917    2           IF CLI$GET_VALUE(QUAL$L_FLUSH,Q_VAL)                        /* Get "FLUSH" string and check status */
2918    2           THEN DO;                                                    /* Value was specified */
2919    3                   Q_VAL.L = INDEX(Q_VAL_SECS,' ') - 1;                /* Eliminate trailing blanks */
2920    3                   IF Q_VAL.L <= 0                                     /* Check for valid size for "seconds" */
2921    3                   THEN DO;
2922    4                        CALL MON_ERR(MNR$_INVFLUSHSP);                 /* Log possible error */
2923    4                        RETURN(MNR$_INVFLUSHSP);                       /* ... and return with status */
2924    4                        END;
2925    3                   CURR_ERRCODE = MNR$_INVFLUSHSP;                     /* Set MONITOR code in case conversion error
2926    3                   M->MRB$L_FLUSH = BIN(SUBSTR(Q_VAL_SECS,1,Q_VAL.L),31);  /* Convert seconds to binary */
2927    3                   CURR_ERRCODE = 0;                                   /* Reset to default MONITOR code */
2928    3                   IF M->MRB$L_FLUSH <= 0                              /* Check for valid value */
2929    3                   THEN DO;
2930    4                        CALL MON_ERR(MNR$_INVFLUSHSP);                 /* Log possible error */
2931    4                        RETURN(MNR$_INVFLUSHSP);                       /* ... and return with status */
2932    4                        END;
2933    3                END;
2934    2
2935    2           ELSE                                                        /* Value was defaulted */
```

```
2936    2           M->MRB$L_FLUSH = DEF_MRBPTR->MRB$L_FLUSH;                   /* Store the default value */
2937    2       END;
2938    1
2939    1   Q_VAL.L = LENGTH(Q_VAL_SECS);                                       /* Set length field of descriptor */
2940    1
2941    1   CALL = CLI$PRESENT(QUAL$L_VIEW);                                    /* Get VIEWING_TIME qualifier presence indic
2942    1   IF CALL = CLI$_PRESENT                                              /* If explicitly present, */
2943    1     THEN
2944    1       DO;
2945    2       QUAL_SPECIFIED = YES;                                           /* Indicate qualifier explicitly specified *
2946    2       IF CLI$GET_VALUE(QUAL$L_VIEW,Q_VAL)                             /* Get "VIEWING_TIME" string and check statu
2947    2       THEN DO;                                                        /* Value was specified */
2948    3               Q_VAL.L = INDEX(Q_VAL_SECS,' ') - 1;                    /* Eliminate trailing blanks */
2949    3               IF Q_VAL.L <= 0                                         /* Check for valid size for "seconds" */
2950    3               THEN DO;
2951    4                    CALL MON_ERR(MNR$_INVVIEWSP);                      /* Log possible error */
2952    4                    RETURN(MNR$_INVVIEWSP);                            /* ... and return with status */
2953    4                    END;
2954    3               CURR_ERRCODE = MNR$_INVVIEWSP;                          /* Set MONITOR code in case conversion error
2955    3               M->MRB$L_VIEWING_TIME = BIN(SUBSTR(Q_VAL_SECS,1,Q_VAL.L),31); /* Convert seconds to binary */
2956    3               CURR_ERRCODE = 0;                                       /* Reset to default MONITOR code */
2957    3               IF M->MRB$L_VIEWING_TIME <= 0                           /* Check for valid value */
2958    3               THEN DO;
2959    4                    CALL MON_ERR(MNR$_INVVIEWSP);                      /* Log possible error */
2960    4                    RETURN(MNR$_INVVIEWSP);                            /* ... and return with status */
2961    4                    END;
2962    3           END;
2963    2
2964    2       ELSE                                                            /* Value was defaulted */
2965    2           M->MRB$L_VIEWING_TIME = DEF_MRBPTR->MRB$L_VIEWING_TIME;     /* Store the default value */
2966    2       END;
2967    1
2968    1
```

```
2969    1       CALL = CLI$PRESENT(QUAL$L_BY_NODE);                     /* Get BY_NODE qualifier presence indicator
2970    1       IF CALL = CLI$_PRESENT                                  /* If explicitly present, */
2971    1         THEN
2972    1           DO;
2973    2           QUAL_SPECIFIED = YES;                               /* Indicate qualifier explicitly specified *
2974    2           M->MRB$V_BY_NODE = YES;                             /* Turn on flag in MRB */
2975    2           END;
2976    1
2977    1       IF CALL = CLI$_NEGATED                                  /* If explicitly negated, */
2978    1           THEN DO;
2979    2                QUAL_SPECIFIED = YES;                          /* Indicate qualifier explicitly specified *
2980    2                M->MRB$V_BY_NODE = NO;                         /* Turn off flag in MRB */
2981    2                END;
2982    1
2983    1       CALL = CLI$PRESENT(QUAL$L_INP);                         /* Get INPUT qualifier presence indicator */
2984    1       IF CALL = CLI$_PRESENT                                  /* If explicitly present, */
2985    1           THEN DO;
2986    2                QUAL_SPECIFIED = YES;                          /* Indicate qualifier explicitly specified *
2987    2                M->MRB$V_MFSUM = NO;                           /* Set to NO for now (may change in BUILD_IF
2988    2                CALL = BUILD_IFB_TABLE();                      /* Build the table of Input File Blocks (IFB
2989    2                IF ^STATUS
2990    2                THEN RETURN(CALL);                             /* return with error status, MON_ERR was alr
2991    2                END;
2992    1       IF CALL = CLI$_NEGATED                                  /* If explicitly negated, */
2993    1           THEN DO;
2994    2                QUAL_SPECIFIED = YES;                          /* Indicate qualifier explicitly specified *
2995    2                M->MRB$V_MFSUM = NO;                           /* Turn off Multi-File Summary indicator */
2996    2                M->MRB$A_INPUT = NULL();                       /* Indicate no input file */
2997    2                END;
2998    1
```

```
2999    1       CALL = CLI$PRESENT(QUAL$L_DISP);                        /* Get DISPLAY qualifier presence indicator
3000    1       IF CALL = CLI$_PRESENT                                  /* If explicitly present, */
3001    1           THEN DO;
3002    2                QUAL_SPECIFIED = YES;                          /* Indicate qualifier explicitly specified *
3003    2                IF DISP_PTR_VOL = NULL()                       /* If no volatile file spec string area, */
3004    2                    THEN DO;
3005    3                         ALLOCATE FILE_SPEC;                   /*   then get one */
3006    3                         DISP_PTR_VOL = FILE_SPEC_PTR;         /*   set up a ptr to it */
3007    3                         FILE_SPEC.L = LENGTH(FILE_SPEC.S);    /*   set length ... */
3008    3                         FILE_SPEC.A = ADDR(FILE_SPEC.S);      /*   and address in descriptor */
3009    3                         END;
3010    2
3011    2                    ELSE DO;
3012    3                         FILE_SPEC_PTR = DISP_PTR_VOL;         /* Otherwise, simply point to existing one *
3013    3                         FILE_SPEC.L = FILE_SPEC_SIZE;         /* ... and re-init its length */
3014    3                         END;
3015    2
3016    2                IF CLI$GET_VALUE(QUAL$L_DISP,FILE_SPEC)        /* Qualifier value specified ? */
3017    2                    THEN DO;
3018    3                         DISP_PTR_SWAP = YES;                  /* Yes -- ind to SET_CMD a ptr swap is neces
3019    3                         TEMP = INDEX(FILE_SPEC.S,' ') - 1;    /* Find trailing blanks in value (string) */
3020    3                         IF TEMP >= 0 THEN FILE_SPEC.L = TEMP; /* If found one, set new length */
3021    3                         M->MRB$A_DISPLAY = FILE_SPEC_PTR;     /* Store away pointer to value descr */
3022    3                         M->MRB$V_DISP_TO_FILE = YES;          /* Indicate filespec specified */
3023    3                         END;
3024    2
3025    2                    ELSE DO;
3026    3                         M->MRB$A_DISPLAY = ADDR(DEF$L_DISP);  /* No -- qualifier value defaulted */
3027    3                         M->MRB$V_DISP_TO_FILE = NO;           /* Store a default value descr */
3028    3                         END;                                  /* ... and default indicator */
3029    2
3030    2                END;
3031    1
3032    1       IF CALL = CLI$_NEGATED                                  /* If explicitly negated, */
3033    1           THEN DO;
3034    2                QUAL_SPECIFIED = YES;                          /* Indicate qualifier explicitly specified *
3035    2                M->MRB$A_DISPLAY = NULL();                     /* Indicate no display output */
3036    2                M->MRB$V_DISP_TO_FILE = NO;                    /* ..... */
3037    2                END;
3038    1
3039    1       CALL = CLI$PRESENT(QUAL$L_REC);                         /* Get RECORD qualifier presence indicator *
3040    1       IF CALL = CLI$_PRESENT                                  /* If explicitly present, */
3041    1           THEN DO;
3042    2                QUAL_SPECIFIED = YES;                          /* Indicate qualifier explicitly specified *
3043    2                IF REC_PTR_VOL = NULL()                        /* If no volatile file spec string area, */
3044    2                    THEN DO;
3045    3                         ALLOCATE FILE_SPEC;                   /*   then get one */
3046    3                         REC_PTR_VOL = FILE_SPEC_PTR;          /*   set up a ptr to it */
3047    3                         FILE_SPEC.L = LENGTH(FILE_SPEC.S);    /*   set length ... */
3048    3                         FILE_SPEC.A = ADDR(FILE_SPEC.S);      /*   and address in descriptor */
3049    3                         END;
3050    2
3051    2                    ELSE FILE_SPEC_PTR = REC_PTR_VOL;          /* Otherwise, simply point to existing one *
3052    2
3053    2                IF CLI$GET_VALUE(QUAL$L_REC,FILE_SPEC)         /* Qualifier value specified ? */
3054    2                    THEN DO;
```

```
3055    3                   REC_PTR_SWAP = YES;                         /* Yes -- ind to SET_CMD a ptr swap is neces
3056    3                   M->MRB$A_RECORD = FILE_SPEC_PTR;            /* Store away pointer to string descr */
3057    3                   END;
3058    2
3059    2               ELSE M->MRB$A_RECORD = ADDR(DEF$L_REC);         /* No -- store a default value */
3060    2
3061    2           END;
3062    1
3063    1       IF CALL = CLI$_NEGATED                                  /* If explicitly negated, */
3064    1           THEN DO;
3065    2                QUAL_SPECIFIED = YES;                          /* Indicate qualifier explicitly specified *
3066    2                M->MRB$A_RECORD = NULL();                      /* Indicate no record output */
3067    2                END;
3068    1
3069    1       CALL = CLI$PRESENT(QUAL$L_SUMM);                        /* Get SUMMARY qualifier presence indicator
3070    1       IF CALL = CLI$_PRESENT                                  /* If explicitly present, */
3071    1           THEN DO;
3072    2                QUAL_SPECIFIED = YES;                          /* Indicate qualifier explicitly specified *
3073    2                IF SUMM_PTR_VOL = NULL()                       /* If no volatile file spec string area, */
3074    2                    THEN DO;
3075    3                         ALLOCATE FILE_SPEC;                   /*   then get one */
3076    3                         SUMM_PTR_VOL = FILE_SPEC_PTR;         /*   set up a ptr to it */
3077    3                         FILE_SPEC.L = LENGTH(FILE_SPEC.S);    /*   set length ... */
3078    3                         FILE_SPEC.A = ADDR(FILE_SPEC.S);      /*   and address in descriptor */
3079    3                         END;
3080    2
3081    2                    ELSE FILE_SPEC_PTR = SUMM_PTR_VOL;         /* Otherwise, simply point to existing one *
3082    2
3083    2                IF CLI$GET_VALUE(QUAL$L_SUMM,FILE_SPEC)        /* Qualifier value specified ? */
3084    2                    THEN DO;
3085    3                         SUMM_PTR_SWAP = YES;                  /* Yes -- ind to SET_CMD a ptr swap is neces
3086    3                         M->MRB$A_SUMMARY = FILE_SPEC_PTR;     /* Store away pointer to string descr */
3087    3                         END;
3088    2
3089    2                    ELSE M->MRB$A_SUMMARY = ADDR(DEF$L_SUMM);  /* No -- store a default value */
3090    2
3091    2                END;
3092    1
3093    1       IF CALL = CLI$_NEGATED                                  /* If explicitly negated, */
3094    1           THEN DO;
3095    2                QUAL_SPECIFIED = YES;                          /* Indicate qualifier explicitly specified *
3096    2                M->MRB$A_SUMMARY = NULL();                     /* Indicate no summary output */
3097    2                END;
3098    1
3099    1       CALL = CLI$PRESENT(QUAL$L_COMM);                        /* Get COMMENT qualifier presence indicator
3100    1       IF CALL = CLI$_PRESENT                                  /* If explicitly present, */
3101    1           THEN DO;
3102    2                QUAL_SPECIFIED = YES;                          /* Indicate qualifier explicitly specified *
3103    2                IF COMM_PTR_VOL = NULL()                       /* If no volatile comment string area, */
3104    2                    THEN DO;
3105    3                         ALLOCATE COMM_STR;                    /*   then get one ... */
3106    3                         COMM_PTR_VOL = COMM_STR_PTR;          /*   and set up a ptr to it */
3107    3                         END;
3108    2
3109    2                    ELSE COMM_STR_PTR = COMM_PTR_VOL;          /* Otherwise, simply point to existing one *
3110    2
3111    2                IF CLI$GET_VALUE(QUAL$L_COMM,DYN_STRING)       /* Qualifier value specified ? */
```

```
3112    2                   THEN DO;
3113    3                        COMM_STR.S = DYN_STRING_S;                 /* Yes -- move string out of dyn area */
3114    3                        COMM_STR.L = DYN_STRING_L;                 /* ... and set up its length */
3115    3                        COMM_STR.A = ADDR(COMM_STR.S);             /* ... and address */
3116    3                        COMM_PTR_SWAP = YES;                       /* Ind to SET_CMD a ptr swap is necessary */
3117    3                        M->MRB$A_COMMENT = COMM_STR_PTR;           /* Store away pointer to string descr */
3118    3                        END;
3119    2
3120    2                   ELSE M->MRB$A_COMMENT = NULL();                 /* No -- store a default value */
3121    2
3122    2               END;
3123    1
3124    1       IF CALL = CLI$_NEGATED                                      /* If explicitly negated, */
3125    1           THEN DO;
3126    2                QUAL_SPECIFIED = YES;                              /* Indicate qualifier explicitly specified */
3127    2                M->MRB$A_COMMENT = NULL();                         /* Indicate no comment string */
3128    2                END;
3129    1
3130    1       RETURN(NORMAL);                                             /* Return with status */
3131    1
```

```
3132    1       BUILD_IFB_TABLE: Procedure Returns(Fixed Binary(31));
3133    2
3134    2       /***
3135    2       /*
3136    2       /* FUNCTIONAL DESCRIPTION:
3137    2       /*
3138    2       /*      BUILD_IFB_TABLE
3139    2       /*
3140    2       /*      This routine builds the IFB (Input File Block) TABLE.
3141    2       /*      In addition, it sets up MRB$A_INPUT to point to the IFB TABLE, and
3142    2       /*      sets up MRB$B_INP_FILES to be the number of input files described by
3143    2       /*      the table.
3144    2       /*
3145    2       /* INPUTS:
3146    2       /*
3147    2       /*      None
3148    2       /*
3149    2       /* IMPLICIT INPUTS:
3150    2       /*
3151    2       /*      IFB_TABLE, MRB
3152    2       /*
3153    2       /* OUTPUTS:
3154    2       /*
3155    2       /*      None
3156    2       /*
3157    2       /* IMPLICIT OUTPUTS:
3158    2       /*
3159    2       /*      IFB_TABLE built, MRB$A_INPUT and MRB$B_INP_FILES estblished.
3160    2       /*
3161    2       /* ROUTINE VALUE:
3162    2       /*
3163    2       /*      Normal, or bad status from LIB$FIND_FILE
3164    2       /*
3165    2       /* SIDE EFFECTS:
3166    2       /*
3167    2       /*      None
3168    2       /*
3169    2       /*/
3170    2
```

```
3171  2     /*
3172  2     /*      !-----------------------------------------------------------------------+
3173  2     /*      !                                                                       !
3174  2     /*      !                        GLOBAL STORAGE DEFINITIONS                     !
3175  2     /*      !                                                                       !
3176  2     /*      !-----------------------------------------------------------------------+
3177  2     /*/
3178  2
3179  2     Declare
3180  2     MAX_INP_FILES          FIXED BINARY(31) GLOBALDEF VALUE INIT(125); /* Max no. of input files for multi-file summary
3181  2
3182  2
3183  2     /*
3184  2     /*      !-----------------------------------------------------------------------+
3185  2     /*      !                                                                       !
3186  2     /*      !                        EXTERNAL REFERENCES                            !
3187  2     /*      !                                                                       !
3188  2     /*      !-----------------------------------------------------------------------+
3189  2     /*/
3190  2
3191  2     %INCLUDE        MONDEF;                                    /* Monitor utility structure definitions */
3959  2
3960  2     Declare
3961  2     MON_ERR         ENTRY (ANY VALUE, ANY, ANY) OPTIONS(VARIABLE),
3962  2                                                        /* Routine to log synchronous errors */
3963  2     DSC$K_DTYPE_T   FIXED BINARY(15) GLOBALREF VALUE,  /* String  descr. type */
3964  2     DSC$K_CLASS_D   FIXED BINARY(15) GLOBALREF VALUE,  /* Dynamic descr. class */
3965  2     SS$_NORMAL      FIXED BINARY(31) GLOBALREF VALUE,  /* System normal return status */
3966  2     RMS$_EOF        FIXED BINARY(31) GLOBALREF VALUE,  /* RMS end-of-file return status */
3967  2     RMS$_NMF        FIXED BINARY(31) GLOBALREF VALUE,  /* RMS no-more-files message for wildcard parsing */
3968  2     RMS$_FNF        FIXED BINARY(31) GLOBALREF VALUE,  /* RMS file-not-found message */
3969  2     MNR$_OPENIN     FIXED BINARY(31) GLOBALREF VALUE,  /* Error message code */
3970  2     MNR$_TOOMNYFILES FIXED BINARY(31) GLOBALREF VALUE, /* Error message code */
3971  2     LIB$FIND_FILE   EXTERNAL ENTRY(ANY,ANY,ANY,ANY,ANY,ANY,ANY)     /* RTL routine to parse a wildcard spec*/
3972  2                     OPTIONS(VARIABLE) RETURNS(FIXED BINARY(31)),
3973  2     LIB$FIND_FILE_END EXTERNAL ENTRY(ANY),             /* RTL routine to end wildcard spec parse*/
3974  2     LIB$SCOPY_DXDX  EXTERNAL ENTRY(ANY,ANY),           /* RTL string copy routine */
3975  2     IFB_TAB_VOL     POINTER GLOBALREF,                 /* Pointer to volatile IFB_TABLE */
3976  2     IFB_TAB_SWAP    BIT(1) ALIGNED GLOBALREF;          /* YES => swap IFB_TAB_VOL and IFB_TAB_PERM */
3977  2
3978  2     /*
3979  2     /*      !-----------------------------------------------------------------------+
3980  2     /*      !                                                                       !
3981  2     /*      !                             LOCAL STORAGE                             !
3982  2     /*      !                                                                       !
3983  2     /*      !-----------------------------------------------------------------------+
3984  2     /*/
3985  2
3986  2     Declare
3987  2     USER_FLAGS      FIXED BINARY(31) INIT(2),          /* Flag indicating "stickiness desired" */
3988  2     CONTEXT         FIXED BINARY(31),
3989  2     CALL            FIXED BINARY(31),                  /* Holds function value (return status) of called ro
3990  2     STATUS          BIT(1) BASED(ADDR(CALL)),          /* Low-order status bit for called routines */
3991  2     FIND_FILE_CALL  FIXED BINARY(31),                  /* Holds function value (return status) of LIB$FIND_
3992  2     FIND_FILE_STAT  BIT(1) BASED(ADDR(FIND_FILE_CALL)), /* Low-order status bit for LIB$FIND_FILE */
3993  2     SPEC_LEN        FIXED BINARY(15),                  /* Input filespec length */
```

```
3994    2       I               FIXED BINARY(7),                        /* Loop control */
3995    2       VALUE_FOUND     BIT(1) ALIGNED,                         /* YES => a value for the /INPUT qualifier found */
3996    2       IFBPTR          POINTER,                                /* Pointer to Input File Block (IFB) */
3997    2       IFB_TAB_PTR     POINTER;                                /* Pointer to IFB_TABLE */
3998    2
3999    2       Declare
4000    2       1 IFB_TABLE     BASED(IFB_TAB_PTR),                     /* Input File Block (IFB) Table */
4001    2         2 AN_IFB      (1:MAX_INP_FILES+1)  CHAR(IFB$K_SIZE);  /* A single IFB */
4002    2
4003    2       Declare
4004    2       REC_DEF_S       CHAR(DEF$L_REC) BASED(DEF$A_REC);       /* String for default recording file spec */
4005    2
```

```
4006    2
4007    2       IF IFB_TAB_VOL = NULL()                                 /* If no volatile IFB table, */
4008    2           THEN DO;                                            /*                           */
4009    3                ALLOCATE IFB_TABLE;                            /*   then get one */
4010    3                IFB_TAB_VOL = IFB_TAB_PTR;                     /*   set up a ptr to it */
4011    3                DO I = 1 TO MAX_INP_FILES + 1;                 /*   clear entire */
4012    4                UNSPEC(AN_IFB(I)) = '0'B;                      /*   array to */
4013    4                END;                                           /*   zeroes */
4014    3                END;
4015    2
4016    2           ELSE IFB_TAB_PTR = IFB_TAB_VOL;                     /* Otherwise, simply point to existing one */
4017    2
4018    2       /*
4019    2       /*      NOTE -- at this point, the volatile pointer (IFB_TAB_VOL) and the base
4020    2       /*              pointer (IFB_TAB_PTR) both point to the IFB_TABLE in use.
4021    2       /*/
4022    2       M->MRB$B_INP_FILES = 0;                                 /* Start off loop with no input files */
4023    2       VALUE_FOUND = YES;                                      /* ... and assume a qualifier value (file spec) found */
4024    2       CALL = SS$_NORMAL;                                      /* Init main loop status */
4025    2       I = 1;                                                  /* Init input file counter */
4026    2       CONTEXT = 0;                                            /* init CONTEXT */
4027    2       /*
4028    2       /*      NOTE -- What follows are two loops - the outer loop does CLI$GET_VALUE calls, and the inner loop does
4029    2       /*              LIB$FIND_FILE calls. The CLI$GET_VALUE loop is controlled by STATUS, which is the low bit of CALL, a
4030    2       /*              also by VALUE_FOUND. CALL will always be SS$_NORMAL unless the LIB$FIND_FILE loop runs into trouble.
4031    2       /*              CALL is what is ultimately returned by this procedure.
4032    2       /*/
4033    2       DO WHILE(STATUS & VALUE_FOUND);                         /* Begin CLI$GET_VALUE loop */
4034    3          FIND_FILE_CALL = SS$_NORMAL;                         /* init FIND_FILE status */
4035    3          DYN_SPEC.T = DSC$K_DTYPE_T;                          /* Init  str. descr. type */
4036    3          DYN_SPEC.C = DSC$K_CLASS_D;                          /* dynamic class */
4037    3          IF ^CLI$GET_VALUE(QUAL$L_INP,DYN_SPEC)               /* File spec specified ? */
4038    3          THEN DO;                                             /* no filespec specified with qualifier */
4039    4               IF M->MRB$B_INP_FILES = 0                       /* If first time around */
4040    4               THEN CALL LIB$SCOPY_DXDX(DEF$L_REC,DYN_SPEC);   /* use default filespec */
4041    4               ELSE VALUE_FOUND = NO;                          /* else indicate no more filespecs to skip LIB$FIND_FILE loo
4042    4               END;
4043    3          DO WHILE(STATUS & FIND_FILE_STAT & VALUE_FOUND);     /* Begin LIB$FIND_FILE loop */
4044    4             IFBPTR = ADDR(AN_IFB(I));                         /* Address an IFB */
4045    4             IF IFB$A_INPUT = NULL()                           /* If not pointing to a file-spec yet, */
4046    4             THEN DO;                                          /*                                     */
4047    5                  ALLOCATE PARSED_SPEC;                        /*   allocate space for result */
4048    5                  IFB$A_INPUT = PARSED_SPEC_PTR;               /*   set up a ptr to it */
4049    5                  PARSED_SPEC.T = DSC$K_DTYPE_T;               /*   str. descr. type */
4050    5                  PARSED_SPEC.C = DSC$K_CLASS_D;               /*   dynamic class */
4051    5                  PARSED_SPEC.A = NULL();                      /*   make sure length, */
4052    5                  PARSED_SPEC.L = 0;                           /*   and address are 0 */
4053    5                  END;
4054    4             ELSE PARSED_SPEC_PTR = IFB$A_INPUT;               /* Otherwise, simply point to existing one */
4055    4             FIND_FILE_CALL=LIB$FIND_FILE(DYN_SPEC,PARSED_SPEC,CONTEXT,DEF$L_REC,,,USER_FLAGS); /* Get the next full file s
/
4056    4             IF FIND_FILE_STAT                                 /* Did we get another valid filespec? */
4057    4                THEN DO;                                       /*                                    */
4058    5                     M->MRB$B_INP_FILES = M->MRB$B_INP_FILES + 1;  /* Yes -- count it */
4059    5                     IF I > MAX_INP_FILES                      /* If we have exceeded the max. allowed # of input f
4060    5                       THEN DO;
```

```
4061    6                       CALL MON_ERR(MNR$_TOOMNYFILES);             /* then log the error */
4062    6                       CALL = MNR$_TOOMNYFILES;                    /* set bad status to end both inner and outer loops
4063    6                       END;
4064    5                       ELSE I = I + 1;                             /*...else increment file count and continue the loop
4065    5                   END;
4066    4               END;                                                /* end of LIB$FIND_FILE loop */
4067    3               IF (^FIND_FILE_STAT & FIND_FILE_CALL ^= RMS$_NMF)   /* "no-more-files" is the only valid error from LIB$
4068    3                   THEN DO;                                        /* ...any other status means trouble (such as file n
4069    4                       CALL MON_ERR(MNR$_OPENIN,FIND_FILE_CALL,PARSED_SPEC);/* log error */
4070    4                       CALL = MNR$_OPENIN;                         /* set bad status to end both the inner and outer lo
4071    4                       END;
4072    3           END;                                                    /* End of CLI$GET_VALUE loop */
4073    2               CALL LIB$FIND_FILE_END(CONTEXT);                    /* wipe out context of prev LIB$FIND_FILE calls */
4074    2           IFB_TAB_SWAP = YES;                                     /* Ind to SET_CMD a ptr swap is necessary */
4075    2           M->MRB$A_INPUT = IFB_TAB_PTR;                           /* Store ptr to IFB table */
4076    2           IF M->MRB$B_INP_FILES > 1  THEN M->MRB$V_MFSUM = YES;   /* If more than 1 file spec, indicate multi-file sum
4077    2
4078    2           RETURN (CALL);                                          /* Return to caller */
4079    2
4080    2           END BUILD_IFB_TABLE;
4081    1
4082    1           END GET_QUALIFIERS;
4083
```

```
4084          GET_CLASSES: Procedure (AT_LEAST_ONE_CLASS) Returns(Fixed Binary(31));
4085    1
4086    1     /***
4087    1     /*
4088    1     /* FUNCTIONAL DESCRIPTION:
4089    1     /*
4090    1     /*      GET_CLASSES
4091    1     /*
4092    1     /*      Communicate with CLE to get class_name keywords; then use keywords as
4093    1     /*      input to LOOKUP_KEY and get back class numbers. Ultimate output is
4094    1     /*      MRBSO_CLASSBITS, a bit string for which each bit number corresponds
4095    1     /*      to class number. If a lookup error is found, bad status is returned
4096    1     /*      to the issuer.
4097    1     /*
4098    1     /* INPUTS:
4099    1     /*
4100    1     /*      TBS
4101    1     /*
4102    1     /* IMPLICIT INPUTS:
4103    1     /*
4104    1     /*      TBS
4105    1     /*
4106    1     /* OUTPUTS:
4107    1     /*
4108    1     /*      TBS
4109    1     /*
4110    1     /* IMPLICIT OUTPUTS:
4111    1     /*
4112    1     /*      TBS
4113    1     /*
4114    1     /* ROUTINE VALUE:
4115    1     /*
4116    1     /*      TBS
4117    1     /*
4118    1     /* SIDE EFFECTS:
4119    1     /*
4120    1     /*      TBS
4121    1     /*
4122    1     /*/
4123    1
```

```
4124  1     /*
4125  1     /*      !-------------------------------------------------------------------------+
4126  1     /*      !                                                                         !
4127  1     /*      !                              LOCAL STORAGE                              !
4128  1     /*      !                                                                         !
4129  1     /*      !-------------------------------------------------------------------------+
4130  1     /*/
4131  1
4132  1     Declare
4133  1     AT_LEAST_ONE_CLASS      BIT(1) ALIGNED,                 /* YES => user requested at least one class */
4134  1     CALL                    FIXED BINARY(31),               /* Holds function value (return status) of called ro
4135  1     STATUS                  BIT(1) BASED(ADDR(CALL)),       /* Low-order status bit "or called routines */
4136  1     CLASS_KEY               FIXED BINARY(31) STATIC,        /* Class keyword number */
4137  1                                                             /* Note -- must be STATIC to get error msg */
4138  1     TEMP                    FIXED BINARY(31);               /* Scratch "register" */
4139  1
4140  1
4141  1     %REPLACE        NOT_SUCCESSFUL          BY '0'B;        /* Failing severity code */
4142  1     %REPLACE        CLASS_TOK_SIZE          BY 20;          /* Size of token for class-name */
4143  1     %REPLACE        OR_OP                   BY '0111'B;     /* OR Boolean operation */
4144  1
4145  1     /*
4146  1     /*      !-------------------------------------------------------------------------+
4147  1     /*      !                                                                         !
4148  1     /*      !                          EXTERNAL REFERENCES                            !
4149  1     /*      !                                                                         !
4150  1     /*      !-------------------------------------------------------------------------+
4151  1     /*/
4152  1
4153  1     %INCLUDE        MONDEF;                                 /* Monitor utility structure definitions */
4921  1
4922  1     Declare
4923  1     CLI$GET_VALUE   ENTRY(ANY, ANY)                         /* CLE routine to get qualifier values */
4924  1                     RETURNS(BIT(1)),
4925  1     LIB$LOOKUP_KEY  EXTERNAL ENTRY(ANY, ANY, FIXED BINARY(31))      /* RTL rtn to look up class-names in a table */
4926  1                     RETURNS(BIT(1)),
4927  1     MON_ERR         ENTRY (ANY VALUE, ANY, ANY) OPTIONS(VARIABLE);  /* MONITOR MACRO-32 routine to log synchronous error
4928  1
4929  1     Declare
4930  1     I               FIXED BINARY(7),                        /* Loop control */
4931  1     MAX_CLASS_NO    FIXED BINARY(31) GLOBALREF VALUE,       /* Maximum defined class number */
4932  1     ALL_CLSNO       FIXED BINARY(31) GLOBALREF VALUE,       /* ALL class Pseudo class number */
4933  1     ALL_CLASS_FOUND BIT(1),                                 /* Flag to indicate ALL class found on command line
4934  1     MNR$_INVCLSNM   FIXED BINARY(31) GLOBALREF VALUE,       /* Error message code */
4935  1     NORMAL          FIXED BINARY(31) GLOBALREF,             /* MONITOR normal return status */
4936  1     CDBPTR          POINTER          GLOBALREF,             /* Pointer to CDB (Class Descriptor Block) */
4937  1     C               POINTER          DEFINED(CDBPTR),       /* Short-hand synonym for CDBPTR */
4938  1     MRBPTR          POINTER          GLOBALREF,             /* Pointer to MRB (Monitor Request Block) */
4939  1     M               POINTER          DEFINED(MRBPTR),       /* Short-hand synonym for MRBPTR */
4940  1     QUALPTR         POINTER          GLOBALREF;             /* Pointer to Qualifier Descriptors Block */
4941  1
4942  1     Declare
4943  1     1 CDBHEAD GLOBALREF,                                    /* Table of CDB's */
4944  1       2 CDBLOCK (0:127) CHAR(CDB$K_SIZE);
4945  1
4946  1     Declare
```

```
4947    1       CLASSTABLE      CHAR(1)         GLOBALREF;                      /* Table of class names & numbers (dummy) */
4948 :  1                                                                       /* Note -- CLASSTABLE is declared here simply */
4949 :  1                                                                       /*   so its address can be referenced below */
4950    1
4951    1       Declare
4952    1       REQUEST_CLASS_MASK      BIT(MAX_CLASS_NO+1),                    /* Requested classes have bits on */
4953    1       REQUEST_CLASS_VEC       (0:MAX_CLASS_NO) BIT(1) DEFINED(REQUEST_CLASS_MASK);
4954 :  1                                                                       /* Alias for REQUEST_CLASS_MASK, but bit-addressable */
4955    1
4956    1       Declare
4957    1       01 CLASS_VAL STATIC,                                            /* String descr for value of CLASS_NAME qualifier */
4958 :  1                                                                       /* Note -- must be STATIC to get INVCLSNM msg */
4959    1           02 L                FIXED BINARY(31),                       /* Length */
4960    1           02 A                POINTER,                                /* Address */
4961    1           02 S                CHAR(CLASS_TOK_SIZE+1);                 /* String */
4962    1
4963    1
4964    1       CLASS_VAL.L = LENGTH(CLASS_VAL.S);                              /* Init length longword of descr */
4965    1       CLASS_VAL.A = ADDR(CLASS_VAL.S);                                /* Init address longword of descr */
4966    1
4967    1       REQUEST_CLASS_MASK = '0'B;                                      /* Turn off all class bits initially */
4968    1       ALL_CLASS_FOUND = NO;                                           /* Assume we won't find ALL pseudo-class on this command lin
4969    1       DO WHILE(CLI$GET_VALUE(QUAL$L_CLASS,CLASS_VAL));                /* Loop once for each requested class */
4970    2           CLASS_VAL.L = INDEX(CLASS_VAL.S,' ') - 1;                   /* Now strip off trailing blanks */
4971    2           IF CLASS_VAL.L < 0 THEN CLASS_VAL.L = CLASS_TOK_SIZE; /* If too long, replace with max token size */
4972    2
4973    2           IF ^ LIB$LOOKUP_KEY(CLASS_VAL,CLASSTABLE,CLASS_KEY) /* Get class keyword number */
4974    2               THEN DO;
4975    3                    CALL MON_ERR(MNR$_INVCLSNM,,CLASS_VAL);            /* Log error if bad class name */
4976    3                    RETURN(MNR$_INVCLSNM);                             /* ... and return with status */
4977    3                    END;
4978    2           CLASS_VAL.L = CLASS_TOK_SIZE + 1;                           /* Restore string len for next loop */
4979    2           IF CLASS_KEY = ALL_CLSNO                                    /* If all classes  */
4980    2               THEN DO;
4981    3                    ALL_CLASS_FOUND = YES;                             /* inidicate we found ALL on this command line */
4982    3                    M->MRB$V_ALL_CLASS = YES;                          /* indicate its ALL class*/
4983    3                       DO I = 0 TO MAX_CLASS_NO;                       /* Loop once for each possible class */
4984    4                       REQUEST_CLASS_VEC(I) = YES;                     /* Turn on bit for this class */
4985    4                       CDBPTR = ADDR(CDBLOCK(I));                      /* Get CDB addressability */
4986    4                       IF C->CDB$V_DISABLE THEN REQUEST_CLASS_VEC(I) = NO; /* If this class disabled, then ignore it */
4987    4                       END;
4988    3                    END;
4989    2               ELSE REQUEST_CLASS_VEC(CLASS_KEY) = YES;                /* Turn on bit for this class */
4990    2           CDBPTR = ADDR(CDBLOCK(CLASS_KEY));                          /* Get CDB addressability */
4991    2           IF C->CDB$V_DISABLE                                         /* If this class disabled, */
4992    2               THEN REQUEST_CLASS_VEC(CLASS_KEY) = NO;                 /*   then ignore it */
4993    2               ELSE DO;                                                /* Otherwise, */
4994    3                    CALL = GET_CLASS_QUALS(CLASS_KEY);                 /* Process class qualifiers for this class */
4995    3                    IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL); /* ... and check status */
4996    3                    END;
4997    2       END;
4998    1
4999    1       IF INDEX(REQUEST_CLASS_MASK,'1'B) ^= 0                          /* If any classes specified, */
5000    1           THEN DO;
5001    2                AT_LEAST_ONE_CLASS = YES;                              /* indicate got at least one */
5002    2                IF ALL_CLASS_FOUND = NO                                /* if we didn't find the ALL pseudo-class */
5003    2                THEN  M->MRB$V_ALL_CLASS = NO;                         /* ...make sure the ALL class flag is clear (this logic is n
```

```
5004  1  2                                        /* for defaulting to work correctly in interactive mode) */
5005     2              M->MRB$O_CLASSBITS = REQUEST_CLASS_MASK;        /* ... move selected bits */
5006     2              END;
5007     1          ELSE DO;                                          /* No classes specified, use MRB$O_CLASSBITS */
5008     2              IF INDEX(M->MRB$O_CLASSBITS,'1'B) = 0                 /* If no classes specified there, */
5009     2                  THEN AT_LEAST_ONE_CLASS = NO;                     /*   then indicate so */
5010     2                  ELSE AT_LEAST_ONE_CLASS = YES;                    /*   else indicate got one or more */
5011     2              END;
5012     1
5013     1      RETURN(NORMAL);                                       /* Return to caller */
5014     1
5015     1
```

```
5016    1       GET_CLASS_QUALS: Procedure (CLSNO) Returns (fixed binary(31));
5017    2
5018    2       Declare
5019    2       CLI$PRESENT     ENTRY(ANY) RETURNS(BIT(1)),             /* CLE routine to determine presence of quals */
5020    2       CLI$GET_VALUE   ENTRY(ANY, ANY, FIXED BINARY(15))       /* CLE routine to get qualifier values */
5021    2                       RETURNS(BIT(1));
5022    2
5023    2       Declare
5024    2       MNR$_QUALINV    FIXED BINARY(31) GLOBALREF VALUE,       /* Error message code */
5025    2       MNR$_SQUALERR   FIXED BINARY(31) GLOBALREF VALUE,       /* Error message code */
5026    2       MNR$_PDQUALERR  FIXED BINARY(31) GLOBALREF VALUE,       /* Error message code */
5027    2       MNR$_INVITEMNM  FIXED BINARY(31) GLOBALREF VALUE;       /* Error message code */
5028    2
5029    2       Declare
5030    2       MODES_CLSNO     FIXED BINARY(31) GLOBALREF VALUE;       /* MODES class number */
5031    2
5032    2       Declare
5033    2       TEMP_CDBPTR             POINTER ;                       /* tmp pointer for ALL class qualifier loop*/
5034    2
5035    2       Declare
5036    2       1 CLASSTABLE  GLOBALREF,                                /* Table of class names & numbers */
5037    2         2 VECTOR_CT   FIXED BINARY(31),                       /* Count of longwords following in table */
5038    2         2 CL_DESCR (0:127),
5039    2           3 CL_PTR    POINTER,                                /* Pointer to class cstring */
5040    2           3 CL_NO     FIXED BINARY(31);                       /* Class number */
5041    2
5042    2       Declare
5043    2       1 STAT_TABLE    GLOBALREF,                              /* Table of pointers to str descrs for statistic qua
5044    2         2 STAT_DESC   (0:STATS-1) POINTER;
5045    2
5046    2       Declare
5047    2       1 PROCD_TABLE   GLOBALREF,                              /* Table of ptrs to str descrs for PROCESSES display
5048    2         2 PROCD_DESC  (0:PROCDISPS-1) POINTER;
5049    2
5050    2       Declare
5051    2       I               FIXED BINARY(7),                        /* Loop control */
5052    2       J               FIXED BINARY(7),                        /* Loop control */
5053    2       QUAL_FOUND      BIT(1) ALIGNED,                         /* NO => haven't seen a qualifier yet */
5054    2       CLSNO           FIXED BINARY(31),                       /* Class number */
5055    2       QD              CHAR(8) BASED,                          /* Dummy qualifier string descr */
5056    2       CLSTR           CHAR(1) BASED;                          /* Dummy first char of class cstring */
5057    2
5058    2       %REPLACE        ITEM_TOK_SIZE           BY 25;          /* Size of token for item name */
5059    2
5060    2       Declare
5061    2       REQUEST_ITEM_MASK       BIT(16),                        /* Requested items have bits on */
5062    2       REQUEST_ITEM_VEC        (0:15) BIT(1) DEFINED(REQUEST_ITEM_MASK);
5063 :  2                                                               /* Alias for REQUEST_ITEM_MASK, but bit-addressable
5064    2
5065    2       Declare
5066    2       01 ITEM_VAL STATIC,                                     /* String descr for value of /ITEM qualifier */
5067 :  2                                                               /* Note -- must be STATIC to get INVITEMNM msg */
5068    2           02 L                FIXED BINARY(31),               /* Length */
5069    2           02 A                POINTER,                        /* Address */
5070    2           02 S                CHAR(ITEM_TOK_SIZE);            /* String */
5071    2
```

```
5072    2       Declare
5073    2       IVAL_LEN                FIXED BINARY(15),               /* Actual length of item value string */
5074    2       ITEM_LTAB               FIXED BINARY(31) BASED,         /* Dummy first longword of item lookup table */
5075    2       ITEM_KEY                FIXED BINARY(31);               /* Item keyword number */
5076  ;   2                                                             /* Note -- must be STATIC to get error msg */
5077    2
5078    2
```

```
5079    2
5080    2       /*
5081    2       /*      NOTE -- CDBPTR (and its synonym C) has been set up by caller.
5082    2       /*/
5083    2
5084    2       /*
5085    2       /*      Check for the presence of each of the statistics qualifiers
5086    2       /*      (ALL, CUR, AVE, MIN, MAX). If specified for the non-standard
5087    2       /*      class (PROCESSES), or if more than one specified, log error
5088    2       /*      and return.
5089    2       /*/
5090    2
5091    2       QUAL_FOUND = NO;                                        /* Indicate no statistics qualifiers found yet */
5092    2
5093    2       DO I = 0 TO STATS-1;                                    /* Loop for each stat qual for this class */
5094    3
5095    3       IF CLI$PRESENT(STAT_DESC(I)->QD)                        /* If this stat qual is present, */
5096    3           THEN IF C->CDB$V_STD = NO                           /* Check if non-standard class */
5097    3                    THEN DO;                                   /* Non-STD -- stat quals not allowed */
5098    4                         CALL MON_ERR(MNR$_QUALINV,,STAT_DESC(I)->QD,CL_PTR(CLSNO)->CLSTR); /* Log an error */
5099    4                         RETURN(MNR$_QUALINV);                                               /* ... and return with status *
5100    4                         END;
5101    3                    ELSE IF QUAL_FOUND = NO                    /* STD class -- If we haven't seen a qual yet, */
5102    3                              THEN DO;
5103    4                                   QUAL_FOUND = YES;         /* Indicate we found one this time */
5104    4                                   IF CLSNO = ALL_CLSNO      /* If ALL classes  */
5105    4                                   THEN DO J = 1 TO MAX_CLASS_NO;  /* Loop once for each possible class */
5106    5                                        IF C->CDB$V_STD = YES      /* If this is a standard class */
5107    5                                        THEN DO;
5108    6                                             TEMP_CDBPTR = ADDR(CDBLOCK(J));       /* ... Get CDB addressability */
5109    6                                             TEMP_CDBPTR->CDB$B_ST = I;            /* ... and move in the requested sta
5110    6                                             TEMP_CDBPTR->CDB$V_EXPLIC = YES;      /* ... also indicate explicit qualif
5111    6                                             END;
5112    5                                        END;                   /* End ALL classes logic */
5113    4                                   C->CDB$B_ST = I;            /* Move in the requested stat code */
5114    4                                   C->CDB$V_EXPLIC = YES;      /* Indicate a class qualifier explicitly specified *
5115    4                                   END;
5116    3                              ELSE DO;                         /* We've seen a stat qual for this class already */
5117    4                                   CALL MON_ERR(MNR$_SQUALERR,,CL_PTR(CLSNO)->CLSTR); /* Log an error */
5118    4                                   RETURN(MNR$_SQUALERR);                             /* Return with status */
5119    4                                   END;
5120    3       END;
5121    2
```

```
5122  2     /*
5123  2     /*      Check for the presence of each of the PROCESSES display qualifiers
5124  2     /*      (TOPCPU, TOPFAULT, TOPDIO, TOPBIO).  If specified for a standard
5125  2     /*      class, or if more than one specified, log error and return.
5126  2     /*/
5127  2
5128  2     QUAL_FOUND = NO;                                        /* Indicate no PROCESSES qualifiers found yet */
5129  2
5130  2     DO I = 0 TO PROCDISPS-1;                                /* Loop for each PROCESSES qualifier */
5131  3
5132  3     IF PROCD_DESC(I) ^= NULL() &                            /* If this proc qual is defined ... */
5133  3        CLI$PRESENT(PROCD_DESC(I)->QD)                       /* ... AND it is present, */
5134  3         THEN IF C->CDB$V_STD = YES                         /* Check if standard class */
5135  3                 THEN DO;                                   /* STD class -- proc quals not allowed */
5136  4                      CALL MON_ERR(MNR$_QUALINV,,PROCD_DESC(I)->QD,CL_PTR(CLSNO)->CLSTR); /* Log an error */
5137  4                      RETURN(MNR$_QUALINV);                                               /* ... and return with status *
5138  4                      END;
5139  3                 ELSE IF QUAL_FOUND = NO                    /* Non-STD class -- If we haven't seen a qual yet, *
5140  3                         THEN DO;
5141  4                              QUAL_FOUND = YES;             /* Indicate we found one this time */
5142  4                              C->CDB$B_ST = I;              /* ... and move in the requested display type */
5143  4                              END;
5144  3                         ELSE DO;                           /* We've seen a proc qual for this class already */
5145  4                              CALL MON_ERR(MNR$_PDQUALERR,,CL_PTR(CLSNO)->CLSTR); /* Log an error */
5146  4                              RETURN(MNR$_PDQUALERR);                             /* Return with status */
5147  4                              END;
5148  3     END;
5149  2
```

```
5150  2      /*
5151  2      /*      Check for the presence of the /ITEM class qualifier. If present,
5152  2      /*      call CLI$GET_VALUE in a loop to obtain the requested item keywords.
5153  2      /*      Then call LIB$LOOKUP_KEY to get numerical bit values for the keywords
5154  2      /*      and set the bits (in CDX$W_IBITS) corresponding to the bit values.
5155  2      /*/
5156  2
5157  2      IF CLI$PRESENT(QUAL$L_ITEM)                                     /* If /ITEM qual is present, */
5158  2          THEN IF C->CDB$V_HOMOG = NO                                 /* Check if homogeneous class */
5159  2                  THEN DO;                                            /* Not homog -- /ITEM qual not allowed */
5160  3                       CALL MON_ERR(MNR$_QUALINV,,QUAL$L_ITEM,CL_PTR(CLSNO)->CLSTR); /* Log an error */
5161  3                       RETURN(MNR$_QUALINV);                          /* ... and return with status */
5162  3                       END;
5163  2
5164  2                  ELSE DO;                                            /* Homog class -- loop picking up items */
5165  3                       ITEM_VAL.L = LENGTH(ITEM_VAL.S);               /* Init length longword of descr */
5166  3                       ITEM_VAL.A = ADDR(ITEM_VAL.S);                 /* Init address longword of descr */
5167  3                       REQUEST_ITEM_MASK = '0'B;                      /* Turn off all item bits initially */
5168  3
5169  3                       DO WHILE(CLI$GET_VALUE(QUAL$L_ITEM,ITEM_VAL,IVAL_LEN)); /* Loop once for each requested item */
5170  4                           ITEM_VAL.L = IVAL_LEN;                     /* Get actual string length */
5171  4
5172  4                           IF ^ LIB$LOOKUP_KEY(ITEM_VAL,C->CDB$A_CDX->CDX$A_ILOOKTAB->ITEM_LTAB,ITEM_KEY)
5173  4                               THEN DO;                               /* Get item keyword number */
5174  5                                    CALL MON_ERR(MNR$_INVITEMNM,,ITEM_VAL);  /* Log error if bad item name */
5175  5                                    RETURN(MNR$_INVITEMNM);           /* ... and return with status */
5176  5                                    END;
5177  4
5178  4                               ELSE DO;                               /* Found value for item keyword */
5179  5                                    REQUEST_ITEM_VEC(ITEM_KEY) = YES; /* Turn on bit for this item */
5180  5                                    ITEM_VAL.L = LENGTH(ITEM_VAL.S); /* Restore string len for next loop */
5181  5                                    END;
5182  4
5183  4                       END;                                           /* End of DO WHILE loop */
5184  3
5185  3                       IF INDEX(REQUEST_ITEM_MASK,'1'B) ^= 0          /* If any items specified, */
5186  3                           THEN C->CDB$A_CDX->CDX$W_IBITS = REQUEST_ITEM_MASK; /* ... move them into the CDX */
5187  3
5188  3                       END;
5189  2
```

```
5190  2      /*
5191  2      /*      The following Begin-End group sets the CDB$V_PERCENT and CDB$V_CPU bits to
5192  2      /*      the proper states, depending on the specifications of the /[NO]PERCENT and
5193  2      /*      /[NO]CPU qualifiers.
5194  2      /*/
5195  2
5196  2      BEGIN;
5197  3      Declare
5198  3      CLI$PRESENT     EXTERNAL ENTRY(ANY) RETURNS(FIXED BINARY(31)),  /* CLE routine to determine presence of qualifiers *
5199  3      CLI$_LOCNEG     FIXED BINARY(31)    GLOBALREF VALUE;            /* CLI$PRESENT return status code for "explicitly ne
5200  3
5201  3      /*
5202  3      /*      Check for presence of /[NO]PERCENT qualifier
5203  3      /*/
5204  3
5205  3      CALL = CLI$PRESENT(QUAL$L_PCENT);                               /* Get "present" indicators for /PERCENT */
5206  3      IF STATUS                                                       /* If present, */
5207  3          THEN IF C->CDB$V_UNIFORM                                    /* If uniform (/PERCENT allowed for unif only) */
5208  3                   THEN C->CDB$V_PERCENT = YES;                       /* ... then indicate percent stats */
5209  3                   ELSE DO;
5210  4                        CALL MON_ERR(MNR$_QUALINV,,QUAL$L_PCENT,CL_PTR(CLSNO)->CLSTR); /* ... otherwise, log an error */
5211  4                        RETURN(MNR$_QUALINV);                                          /* ... and return with status */
5212  4                        END;
5213  3          ELSE IF CALL = CLI$_LOCNEG THEN C->CDB$V_PERCENT = NO;      /* IF /PERCENT explicitly negated, turn it off */
5214  3
5215  3      /*
5216  3      /*      Now check for presence of /[NO]CPU qualifier
5217  3      /*/
5218  3
5219  3      CALL = CLI$PRESENT(QUAL$L_CPU);                                 /* Get "present" indicators for /CPU */
5220  3      IF STATUS                                                       /* If present, */
5221  3          THEN IF CLSNO = MODES_CLSNO                                 /* If MODES class, */
5222  3                   THEN C->CDB$V_CPU = YES;                           /* ... indicate CPU-specific display */
5223  3                   ELSE DO;                                           /* Otherwise, /CPU is not allowed */
5224  4                        CALL MON_ERR(MNR$_QUALINV,,QUAL$L_CPU,CL_PTR(CLSNO)->CLSTR); /* Log an error */
5225  4                        RETURN(MNR$_QUALINV);                                        /* ... and return with status */
5226  4                        END;
5227  3          ELSE IF CALL = CLI$_LOCNEG THEN C->CDB$V_CPU = NO;          /* IF /CPU explicitly negated, turn it off */
5228  3
5229  3      END;                                                            /* Terminate Begin-End group */
5230  2
5231  2      RETURN(NORMAL);
5232  2
5233  2      END GET_CLASS_QUALS;
5234  1
5235  1      END GET_CLASSES;
5236
```

```
5237          MONITOR_CMD: Procedure Returns (Fixed Binary(31));          /* Routine to execute a MONITOR subcommand */
5238    1
5239    1     /***
5240    1     /*
5241    1     /* FUNCTIONAL DESCRIPTION:
5242    1     /*
5243    1     /*      MONITOR_CMD
5244    1     /*
5245    1     /*      TBS
5246    1     /*
5247    1     /* INPUTS:
5248    1     /*
5249    1     /*      TBS
5250    1     /*
5251    1     /* IMPLICIT INPUTS:
5252    1     /*
5253    1     /*      TBS
5254    1     /*
5255    1     /* OUTPUTS:
5256    1     /*
5257    1     /*      TBS
5258    1     /*
5259    1     /* IMPLICIT OUTPUTS:
5260    1     /*
5261    1     /*      TBS
5262    1     /*
5263    1     /* ROUTINE VALUE:
5264    1     /*
5265    1     /*      TBS
5266    1     /*
5267    1     /* SIDE EFFECTS:
5268    1     /*
5269    1     /*      TBS
5270    1     /*
5271    1     '*/
5272    1
```

```
5273  1     /*
5274  1     /*      !-----------------------------------------------------------------------!
5275  1     /*      !                                                                       !
5276  1     /*      !                            LOCAL STORAGE                              !
5277  1     /*      !                                                                       !
5278  1     /*      !-----------------------------------------------------------------------!
5279  1     /*/
5280  1
5281  1     Declare
5282  1       CALL                  FIXED BINARY(31),                 /* Holds function value (return status) of called ro
5283  1       STATUS                BIT(1) BASED(ADDR(CALL)),         /* Low-order status bit for called routines */
5284  1       AT_LEAST_ONE_CLASS    BIT(1) ALIGNED,                   /* YES => user requested at least one class */
5285  1       IFBPTR                POINTER;                          /* Pointer to Input File Block (IFB) */
5286  1
5287  1
5288  1     %REPLACE                NOT_SUCCESSFUL          BY '0'B;  /* Failing severity code */
5289  1
5290  1     /*
5291  1     /*      !-----------------------------------------------------------------------!
5292  1     /*      !                                                                       !
5293  1     /*      !                         EXTERNAL REFERENCES                           !
5294  1     /*      !                                                                       !
5295  1     /*      !-----------------------------------------------------------------------!
5296  1     /*/
5297  1
5298  1     %INCLUDE        MONDEF;                                   /* Monitor utility structure definitions */
6066  1
6067  1     Declare
6068  1     MON_ERR         ENTRY (ANY VALUE, ANY, ANY) OPTIONS(VARIABLE),  /* MONITOR MACRO-32 routine to log synchronous error
6069  1     MOVE_CLASS_QUALS ENTRY (FIXED BINARY(7))                  /* MACRO-32 routine to move class ... */
6070  1                      RETURNS(FIXED BINARY(31)),               /* ... qualifier values for all classes */
6071  1     GET_CLASSES     ENTRY (BIT(1) ALIGNED) RETURNS(FIXED BINARY(31)), /* Routine to get info on all classes for this req
6072  1     GET_QUALIFIERS  ENTRY RETURNS(FIXED BINARY(31)),          /* Routine to store qualifier info for this request
6073  1     EXECUTE_REQUEST ENTRY RETURNS(FIXED BINARY(31)),          /* Routine to execute a single MONITOR request */
6074  1     MFSUM_REQUEST   ENTRY RETURNS(FIXED BINARY(31));          /* Routine to execute a single MONITOR request with
6075  1
6076  1     Declare
6077  1       MNR$_NOCLASSES FIXED BINARY(31) GLOBALREF VALUE,        /* Message code */
6078  1       CUR_TO_ACT     FIXED BINARY(31) GLOBALREF VALUE,        /* Code value for MOVE_CLASS_QUALS rtn */
6079  1       ALL_TO_ACT     FIXED BINARY(31) GLOBALREF VALUE,        /* Code value for MOVE_CLASS_QUALS rtn */
6080  1       MRBPTR         POINTER GLOBALREF,                       /* Pointer to MRB (Monitor Request Block) */
6081  1       M              POINTER DEFINED(MRBPTR),                 /* Short-hand synonym for MRBPTR */
6082  1       CURR_MRBPTR    POINTER GLOBALREF,                       /* Pointer to "current" MRB (Monitor Request Block)
6083  1       ACT_MRBPTR     POINTER GLOBALREF,                       /* Pointer to "active" MRB (Monitor Request Block) *
6084  1       FIRST_MON_CMD  BIT(1) ALIGNED GLOBALREF,                /* YES => first MONITOR (DCL-level) cmd executing */
6085  1       QUAL_SPECIFIED BIT(1) ALIGNED GLOBALREF,                /* YES => at least 1 qualifier explicitly specified
6086  1       PROMPT         BIT(1) ALIGNED GLOBALREF;                /* YES => prompt user for another subcommand */
6087  1
```

```
6088    1       CALL = MOVE_CLASS_QUALS(CUR_TO_ACT);                    /* Move current class qual values to active */
6089    1       IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL);          /* Return if bad status */
6090    1       ACT_MRBPTR->MRB = CURR_MRBPTR->MRB;                     /* Move current MRB to active MRB */
6091    1       MRBPTR = ACT_MRBPTR;                                    /* Make all MRB refs refer to "active" MRB */
6092    1       CALL = GET_CLASSES(AT_LEAST_ONE_CLASS);                 /* Get info on requested classes */
6093    1       IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL);          /* Return if bad status */
6094    1       CALL = GET_QUALIFIERS();                                /* Get the MONITOR command qualifiers */
6095    1       IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL);          /* Return if bad status */
6096    1
6097    1       IF AT_LEAST_ONE_CLASS                                   /* If at least one class requested, */
6098    1           THEN
6099    1                IF M->MRB$V_MFSUM                              /* If this is a multi-file summary request, */
6100    1                    THEN CALL = MFSUM_REQUEST();               /* Call special REQUEST routine for m.f. summary */
6101    1                    ELSE DO;                                   /* Regular MONITOR request */
6102    2                         IF M->MRB$A_INPUT ^= NULL()           /* If it's a playback, */
6103    2                             THEN DO;
6104    3                                  IFBPTR = M->MRB$A_INPUT;     /* Make MRB$A_INPUT point to a */
6105    3                                  M->MRB$A_INPUT = IFB$A_INPUT;/* ... file descr instead of an IFB */
6106    3                                  END;
6107    2
6108    2                         IF M->MRB$A_SUMMARY ^= NULL()         /* If summary requested, */
6109    2                             THEN DO;
6110    3                                  CALL = MOVE_CLASS_QUALS(ALL_TO_ACT);          /* Force ALL stat value to active */
6111    3                                  IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL); /* Return if bad status */
6112    3                                  END;
6113    2
6114    2                         CALL = EXECUTE_REQUEST();             /* Execute the MONITOR request */
6115    2                         END;
6116    1           ELSE                                                /* Otherwise, no classes requested, */
6117    1                IF FIRST_MON_CMD = YES & QUAL_SPECIFIED = NO   /* Check for bare MONITOR cmd at DCL */
6118    1                    THEN PROMPT = YES;                         /* If so, user wants to go interactive */
6119    1                    ELSE DO;                                   /* MONITOR cmd issued without classes */
6120    2                         CALL MON_ERR(MNR$_NOCLASSES);         /* This is an error ... log it */
6121    2                         RETURN(MNR$_NOCLASSES);               /* ... and return */
6122    2                         END;
6123    1       RETURN (CALL);                                          /* Return with status code */
6124    1
6125    1       END MONITOR_CMD;
6126
```

```
6127          SET_CMD: Procedure Returns (Fixed Binary(31));          /* Routine to execute a SET subcommand */
6128    1
6129    1     /***
6130    1     /*
6131    1     /* FUNCTIONAL DESCRIPTION:
6132    1     /*
6133    1     /*      SET_CMD
6134    1     /*
6135    1     /*      TBS
6136    1     /*
6137    1     /* INPUTS:
6138    1     /*
6139    1     /*      TBS
6140    1     /*
6141    1     /* IMPLICIT INPUTS:
6142    1     /*
6143    1     /*      TBS
6144    1     /*
6145    1     /* OUTPUTS:
6146    1     /*
6147    1     /*      TBS
6148    1     /*
6149    1     /* IMPLICIT OUTPUTS:
6150    1     /*
6151    1     /*      TBS
6152    1     /*
6153    1     /* ROUTINE VALUE:
6154    1     /*
6155    1     /*      TBS
6156    1     /*
6157    1     /* SIDE EFFECTS:
6158    1     /*
6159    1     /*      TBS
6160    1     /*
6161    1     /*/
6162    1
```

```
6163    1
6164    1       %INCLUDE        MONDEF;                                         /* Monitor utility structure definitions */
6932    1
6933    1       %REPLACE                NOT_SUCCESSFUL          BY '0'B;        /* Failing severity code */
6934    1
6935    1       /*
6936    1       /*      !-------------------------------------------------------------------------+
6937    1       /*      !                                                                         !
6938    1       /*      !                       EXTERNAL STORAGE  DEFINITIONS                     !
6939    1       /*      !                                                                         !
6940    1       /*      !-------------------------------------------------------------------------+
6941    1       /*/
6942    1
6943    1       Declare
6944    1       INP_PTR_VOL     POINTER GLOBALDEF       INIT(NULL()),           /* Pointer to volatile /INPUT file-spec */
6945    1       DISP_PTR_VOL    POINTER GLOBALDEF       INIT(NULL()),           /* Pointer to volatile /DISPLAY file-spec */
6946    1       REC_PTR_VOL     POINTER GLOBALDEF       INIT(NULL()),           /* Pointer to volatile /RECORD file-spec */
6947    1       SUMM_PTR_VOL    POINTER GLOBALDEF       INIT(NULL()),           /* Pointer to volatile /SUMMARY file-spec */
6948    1       COMM_PTR_VOL    POINTER GLOBALDEF       INIT(NULL()),           /* Pointer to volatile /COMMENT string */
6949    1       IFB_TAB_VOL     POINTER GLOBALDEF       INIT(NULL());           /* Pointer to volatile IFB_TABLE */
6950    1
6951    1       Declare
6952    1       INP_PTR_SWAP    BIT(1) ALIGNED GLOBALDEF,                       /* YES => swap INP_PTR_VOL and INP_PTR_PERM */
6953    1       DISP_PTR_SWAP   BIT(1) ALIGNED GLOBALDEF,                       /* YES => swap DISP_PTR_VOL and DISP_PTR_PERM */
6954    1       REC_PTR_SWAP    BIT(1) ALIGNED GLOBALDEF,                       /* YES => swap REC_PTR_VOL and REC_PTR_PERM */
6955    1       SUMM_PTR_SWAP   BIT(1) ALIGNED GLOBALDEF,                       /* YES => swap SUMM_PTR_VOL and SUMM_PTR_PERM */
6956    1       COMM_PTR_SWAP   BIT(1) ALIGNED GLOBALDEF,                       /* YES => swap COMM_PTR_VOL and COMM_PTR_PERM */
6957    1       IFB_TAB_SWAP    BIT(1) ALIGNED GLOBALDEF;                       /* YES => swap IFB_TAB_VOL and IFB_TAB_PERM */
6958    1
6959    1       /*
6960    1       /*      !-------------------------------------------------------------------------+
6961    1       /*      !                                                                         !
6962    1       /*      !                           EXTERNAL REFERENCES                           !
6963    1       /*      !                                                                         !
6964    1       /*      !-------------------------------------------------------------------------+
6965    1       /*/
6966    1
6967    1       Declare
6968    1       MOVE_CLASS_QUALS ENTRY (FIXED BINARY(7))                        /* MACRO-32 routine to move class ... */
6969    1                        RETURNS(FIXED BINARY(31)),                     /* ... qualifier values for all classes */
6970    1       GET_CLASSES     ENTRY (BIT(1) ALIGNED) RETURNS(FIXED BINARY(31)), /* Routine to get info on all classes for this req
6971    1       GET_QUALIFIERS  ENTRY RETURNS(FIXED BINARY(31));                /* Routine to store qualifier info for this request
6972    1
6973    1       Declare
6974    1         CUR_TO_ACT    FIXED BINARY(31) GLOBALREF VALUE,               /* Code value for MOVE_CLASS_QUALS rtn */
6975    1         ACT_TO_CUR    FIXED BINARY(31) GLOBALREF VALUE,               /* Code value for MOVE_CLASS_QUALS rtn */
6976    1         MRBPTR        POINTER GLOBALREF,                              /* Pointer to MRB (Monitor Request Block) */
6977    1         CURR_MRBPTR   POINTER GLOBALREF,                              /* Pointer to "current" MRB (Monitor Request Block)
6978    1         TEMP_MRBPTR   POINTER GLOBALREF;                              /* Pointer to "temp" MRB (Monitor Request Block) */
6979    1
6980    1       /*
6981    1       /*      !-------------------------------------------------------------------------+
6982    1       /*      !                                                                         !
6983    1       /*      !                               LOCAL STORAGE                             !
6984    1       /*      !                                                                         !
6985    1       /*      !-------------------------------------------------------------------------+
```

```
6986 ;  1       /*/
6987    1
6988    1       Declare
6989    1       INP_PTR_PERM     POINTER STATIC          INIT(NULL()),          /* Pointer to permanent /INPUT file-spec */
6990    1       DISP_PTR_PERM    POINTER STATIC          INIT(NULL()),          /* Pointer to permanent /DISPLAY file-spec */
6991    1       REC_PTR_PERM     POINTER STATIC          INIT(NULL()),          /* Pointer to permanent /RECORD file-spec */
6992    1       SUMM_PTR_PERM    POINTER STATIC          INIT(NULL()),          /* Pointer to permanent /SUMMARY file-spec */
6993    1       COMM_PTR_PERM    POINTER STATIC          INIT(NULL()),          /* Pointer to permanent /COMMENT string */
6994    1       IFB_TAB_PERM     POINTER STATIC          INIT(NULL());          /* Pointer to permanent IFB_TABLE */
6995    1
6996    1       Declare
6997    1         CALL                   FIXED BINARY(31),                     /* Holds function value (return status) of called ro
6998    1         STATUS                 BIT(1) BASED(ADDR(CALL)),             /* Low-order status bit for called routines */
6999    1         TEMP                   POINTER,                              /* Temporary "scratch" area */
7000    1         AT_LEAST_ONE_CLASS     BIT(1) ALIGNED;                       /* YES => user requested at least one class */
7001    1
7002    1
```

```
7003    1       INP_PTR_SWAP = NO;                              /* Ind no swap needed for 2 ptrs to /INPUT file-spec
7004    1       DISP_PTR_SWAP = NO;                             /* Ind no swap needed for 2 ptrs to /DISPLAY file-sp
7005    1       REC_PTR_SWAP = NO;                              /* Ind no swap needed for 2 ptrs to /RECORD file-spe
7006    1       SUMM_PTR_SWAP = NO;                             /* Ind no swap needed for 2 ptrs to /SUMMARY file-sp
7007    1       COMM_PTR_SWAP = NO;                             /* Ind no swap needed for 2 ptrs to /COMMENT string
7008    1       IFB_TAB_SWAP = NO;                              /* Ind no swap needed for 2 ptrs to IFB_TABLE */
7009    1       TEMP_MRBPTR->MRB = CURR_MRBPTR->MRB;            /* Move "current" MRB to "temp" MRB */
7010    1       MRBPTR = TEMP_MRBPTR;                           /* Make all MRB refs refer to "temp" MRB */
7011    1       CALL = GET_QUALIFIERS();                        /* Get SET qualifiers */
7012    1       IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL);  /* Return if bad status */
7013    1
7014    1       CALL = MOVE_CLASS_QUALS(CUR_TO_ACT);            /* Move current class qual values to active */
7015    1       IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL);  /* Return if bad status */
7016    1
7017    1       CALL = GET_CLASSES(AT_LEAST_ONE_CLASS);         /* Get info on requested classes */
7018    1
7019    1       IF STATUS                                       /* If successful, */
7020    1           THEN DO;
7021    2                TEMP = TEMP_MRBPTR;                    /* Swap "current" and ... */
7022    2                TEMP_MRBPTR = CURR_MRBPTR;             /* ... "temp" MRB pointers */
7023    2                CURR_MRBPTR = TEMP;                    /* .... */
7024    2
7025    2                IF INP_PTR_SWAP = YES                  /* If pointer swap required, */
7026    2                    THEN DO;
7027    3                         TEMP = INP_PTR_VOL;           /* Swap volatile and ... */
7028    3                         INP_PTR_VOL = INP_PTR_PERM;   /* ... permanent /INPUT file-spec */
7029    3                         INP_PTR_PERM = TEMP;          /* ... pointers */
7030    3                         END;
7031    2
7032    2                IF DISP_PTR_SWAP = YES                 /* If pointer swap required, */
7033    2                    THEN DO;
7034    3                         TEMP = DISP_PTR_VOL;          /* Swap volatile and ... */
7035    3                         DISP_PTR_VOL = DISP_PTR_PERM; /* ... permanent /DISPLAY file-spec */
7036    3                         DISP_PTR_PERM = TEMP;         /* ... pointers */
7037    3                         END;
7038    2
7039    2                IF REC_PTR_SWAP = YES                  /* If pointer swap required, */
7040    2                    THEN DO;
7041    3                         TEMP = REC_PTR_VOL;           /* Swap volatile and ... */
7042    3                         REC_PTR_VOL = REC_PTR_PERM;   /* ... permanent /RECORD file-spec */
7043    3                         REC_PTR_PERM = TEMP;          /* ... pointers */
7044    3                         END;
7045    2
7046    2                IF SUMM_PTR_SWAP = YES                 /* If pointer swap required, */
7047    2                    THEN DO;
7048    3                         TEMP = SUMM_PTR_VOL;          /* Swap volatile and ... */
7049    3                         SUMM_PTR_VOL = SUMM_PTR_PERM; /* ... permanent /SUMMARY file-spec */
7050    3                         SUMM_PTR_PERM = TEMP;         /* ... pointers */
7051    3                         END;
7052    2
7053    2                IF COMM_PTR_SWAP = YES                 /* If pointer swap required, */
7054    2                    THEN DO;
7055    3                         TEMP = COMM_PTR_VOL;          /* Swap volatile and ... */
7056    3                         COMM_PTR_VOL = COMM_PTR_PERM; /* ... permanent /COMMENT string */
7057    3                         COMM_PTR_PERM = TEMP;         /* ... pointers */
7058    3                         END;
```

```
7059     2
7060     2              IF IFB_TAB_SWAP = YES                                  /* If pointer swap required, */
7061     2                  THEN DO;
7062     3                       TEMP = IFB_TAB_VOL;                           /* Swap volatile and ... */
7063     3                       IFB_TAB_VOL = IFB_TAB_PERM;                   /* ... permanent IFB_TABLE */
7064     3                       IFB_TAB_PERM = TEMP;                          /* ... pointers */
7065     3                       END;
7066     2
7067     2              CALL = MOVE_CLASS_QUALS(ACT_TO_CUR);                   /* Move active class qual values back to current */
7068     2              IF STATUS = NOT_SUCCESSFUL THEN RETURN (CALL);         /* Return if bad status */
7069     2
7070     2              END;
7071     1
7072     1      RETURN (CALL);                                                 /* Return with status code */
7073     1
7074     1      END SET_CMD;
7075
```

```
7076          INIT_CMD: Procedure Returns (Fixed Binary(31));          /* Routine to execute an INITIALIZE subcommand */
7077    1
7078    1     /***
7079    1     /*
7080    1     /* FUNCTIONAL DESCRIPTION:
7081    1     /*
7082    1     /*      INIT_CMD
7083    1     /*
7084    1     /*      TBS
7085    1     /*
7086    1     /* INPUTS:
7087    1     /*
7088    1     /*      TBS
7089    1     /*
7090    1     /* IMPLICIT INPUTS:
7091    1     /*
7092    1     /*      TBS
7093    1     /*
7094    1     /* OUTPUTS:
7095    1     /*
7096    1     /*      TBS
7097    1     /*
7098    1     /* IMPLICIT OUTPUTS:
7099    1     /*
7100    1     /*      TBS
7101    1     /*
7102    1     /* ROUTINE VALUE:
7103    1     /*
7104    1     /*      TBS
7105    1     /*
7106    1     /* SIDE EFFECTS:
7107    1     /*
7108    1     /*      TBS
7109    1     /*
7110    1     /*/
7111    1
```

```
7112  1     /*
7113  1     /*      ,-------------------------------------------------------------------------.
7114  1     /*      |                                                                         |
7115  1     /*      |                              LOCAL STORAGE                              |
7116  1     /*      |                                                                         |
7117  1     /*      `-------------------------------------------------------------------------'
7118  1     /*/
7119  1
7120  1     Declare
7121  1       CALL                  FIXED BINARY(31);                 /* Holds function value (return status) of called ro
7122  1
7123  1
7124  1     /*
7125  1     /*      ,-------------------------------------------------------------------------.
7126  1     /*      |                                                                         |
7127  1     /*      |                          EXTERNAL REFERENCES                            |
7128  1     /*      |                                                                         |
7129  1     /*      `-------------------------------------------------------------------------'
7130  1     /*/
7131  1
7132  1     %INCLUDE        MONDEF;                                   /* Monitor utility structure definitions */
7900  1
7901  1     Declare
7902  1     DEF_TO_CUR      FIXED BINARY(31) GLOBALREF VALUE;         /* Code value for MOVE_CLASS_QUALS rtn */
7903  1
7904  1     Declare
7905  1     MOVE_CLASS_QUALS        ENTRY (FIXED BINARY(7))           /* MACRO-32 routine to move class ... */
7906  1                             RETURNS(FIXED BINARY(31));        /* ... qualifier values for all classes */
7907  1
7908  1     Declare
7909  1       CURR_MRBPTR   POINTER GLOBALREF,                        /* Pointer to "current" MRB (Monitor Request Block)
7910  1       DEF_MRBPTR    POINTER GLOBALREF;                        /* Pointer to "default" MRB (Monitor Request Block)
7911  1
```

```
7912    1       CURR_MRBPTR->MRB = DEF_MRBPTR->MRB;             /* Move default MRB to current MRB */
7913    1       CALL = MOVE_CLASS_QUALS(DEF_TO_CUR);            /* ... and default class qual values to current */
7914    1
7915    1       RETURN (CALL);                                  /* Return with status code from MOVE_CLASS_QUALS */
7916    1
7917    1       END INIT_CMD;
7918
```

```
7919          HELP_CMD: Procedure Returns (Fixed Binary(31));                 /* Routine to execute a HELP subcommand */
7920    1
7921    1     /***
7922    1     /*
7923    1     /* FUNCTIONAL DESCRIPTION:
7924    1     /*
7925    1     /*      HELP_CMD
7926    1     /*
7927    1     /*      TBS
7928    1     /*
7929    1     /* INPUTS:
7930    1     /*
7931    1     /*      TBS
7932    1     /*
7933    1     /* IMPLICIT INPUTS:
7934    1     /*
7935    1     /*      TBS
7936    1     /*
7937    1     /* OUTPUTS:
7938    1     /*
7939    1     /*      TBS
7940    1     /*
7941    1     /* IMPLICIT OUTPUTS:
7942    1     /*
7943    1     /*      TBS
7944    1     /*
7945    1     /* ROUTINE VALUE:
7946    1     /*
7947    1     /*      TBS
7948    1     /*
7949    1     /* SIDE EFFECTS:
7950    1     /*
7951    1     /*      TBS
7952    1     /*
7953    1     /*/
7954    1
```

```
7955    1       %INCLUDE                $HLPDEF;                                /* HELP flag definitions */
7992    1       %REPLACE                NOT_SUCCESSFUL          BY '0'B;        /* Failing severity code */
7993    1       %REPLACE                MON_HELPLIB             BY 'MNRHELP';   /* Log name for MONITOR subcommand help library */
7994    1       %REPLACE                HELP_PARM               BY 'HELP_KEYS'; /* Name of parameter for HELP command (rest of line)
7995    1
7996    1       /*
7997    1       /*      +-------------------------------------------------------------------------+
7998    1       /*      |                                                                         |
7999    1       /*      |                              LOCAL STORAGE                              |
8000    1       /*      |                                                                         |
8001    1       /*      +-------------------------------------------------------------------------+
8002    1       /*/
8003    1       Declare
8004    1         CALL                  FIXED BINARY(31),                       /* Holds function value (return status) of called ro
8005    1         STATUS                BIT(1) BASED(ADDR(CALL));               /* Low-order status bit for called routines */
8006    1
8007    1       /*
8008    1       /*      +-------------------------------------------------------------------------+
8009    1       /*      |                                                                         |
8010    1       /*      |                          EXTERNAL REFERENCES                            |
8011    1       /*      |                                                                         |
8012    1       /*      +-------------------------------------------------------------------------+
8013    1       /*/
8014    1
8015    1       Declare
8016    1       CLI$GET_VALUE   ENTRY (CHAR(*), ANY)                            /* CLE routine to get qualifier values */
8017    1                       RETURNS(BIT(1)),
8018    1       LBR$OUTPUT_HELP ENTRY (ENTRY VALUE, ANY, ANY, CHAR(*), FIXED BINARY(31), ENTRY VALUE)
8019    1                       OPTIONS(VARIABLE) RETURNS(FIXED BINARY(31)),    /* Rtn to output help info with prompting */
8020    1       LIB$PUT_OUTPUT  ENTRY,                                          /* Routine to write a line to terminal */
8021    1       LIB$GET_INPUT   ENTRY,                                          /* Routine to read terminal */
8022    1       MON_ERR         ENTRY (ANY VALUE, ANY, ANY) OPTIONS(VARIABLE),
8023    1                                                                       /* Routine to log synchronous errors */
8024    1       MNR$_HELPERR    FIXED BINARY(31) GLOBALREF VALUE;               /* Error message code */
8025    1
8026    1       Declare
8027    1       NORMAL          FIXED BINARY(31) GLOBALREF,                     /* MONITOR normal return status */
8028    1       CURR_ERRCODE    FIXED BINARY(31) GLOBALREF;                     /* MONITOR error status code currently expected */
8029    1
8030    1       Declare
8031    1       1 DYN_STRING    GLOBALREF,                                      /* Dynamic string descriptor */
8032    1         2 L           FIXED BINARY(15),                               /* Length */
8033    1         2 TC          CHAR(2),                                        /* Type and Class */
8034    1         2 A           POINTER;                                        /* Address */
8035    1
8036    1
```

```
8037     1
8038     1      CURR_ERRCODE = MNR$_HELPERR;                              /* Set MONITOR code for signaled errors */
8039     1      IF ^CLI$GET_VALUE(HELP_PARM,DYN_STRING)                   /* Get all command line after "HELP" verb */
8040   : 1                                                                 /* ... then ship it off to LBR$OUTPUT_HELP */
8041     1          THEN DYN_STRING.L = 0;                                /* If rest of line null, make string null */
8042     1
8043     1      CALL = LBR$OUTPUT_HELP(LIB$PUT_OUTPUT,,DYN_STRING,MON_HELPLIB,
8044     1                             HLP$M_PROMPT+HLP$M_PROCESS+HLP$M_GROUP+HLP$M_SYSTEM+HLP$M_HELP,
8045     1                             LIB$GET_INPUT);                    /* Call standard HELP output routine */
8046     1
8047     1      IF STATUS = NOT_SUCCESSFUL                                /* If failed, */
8048     1          THEN DO;
8049     2               CALL MON_ERR(MNR$_HELPERR,CALL);                 /* Log the error ... */
8050     2               RETURN(MNR$_HELPERR);                            /* ... and return with status */
8051     2               END;
8052     1
8053     1      RETURN (NORMAL);                                          /* Return with success code */
8054     1
8055     1      END HELP_CMD;
8056
```

```
8057          EXIT_CMD: Procedure Returns (Fixed Binary(31));                 /* Routine to execute an EXIT subcommand */
8058    1
8059    1     /***
8060    1     /*
8061    1     /* FUNCTIONAL DESCRIPTION:
8062    1     /*
8063    1     /*      EXIT_CMD
8064    1     /*
8065    1     /*      TBS
8066    1     /*
8067    1     /* INPUTS:
8068    1     /*
8069    1     /*      TBS
8070    1     /*
8071    1     /* IMPLICIT INPUTS:
8072    1     /*
8073    1     /*      TBS
8074    1     /*
8075    1     /* OUTPUTS:
8076    1     /*
8077    1     /*      TBS
8078    1     /*
8079    1     /* IMPLICIT OUTPUTS:
8080    1     /*
8081    1     /*      TBS
8082    1     /*
8083    1     /* ROUTINE VALUE:
8084    1     /*
8085    1     /*      TBS
8086    1     /*
8087    1     /* SIDE EFFECTS:
8088    1     /*
8089    1     /*      TBS
8090    1     /*
8091    1     /*/
8092    1
```

```
8093  1       /*
8094  1       /*      +-----------------------------------------------------------------------+
8095  1       /*      |                                                                       |
8096  1       /*      |                          LOCAL STORAGE                                |
8097  1       /*      |                                                                       |
8098  1       /*      +-----------------------------------------------------------------------+
8099  1       /*/
8100  1
8101  1
8102  1
8103  1       /*
8104  1       /*      +-----------------------------------------------------------------------+
8105  1       /*      |                                                                       |
8106  1       /*      |                       EXTERNAL REFERENCES                             |
8107  1       /*      |                                                                       |
8108  1       /*      +-----------------------------------------------------------------------+
8109  1       /*/
8110  1
8111  1       Declare
8112  1         PROMPT        BIT(1) ALIGNED  GLOBALREF,          /* YES => prompt user for another subcommand */
8113  1         NORMAL        FIXED BINARY(31) GLOBALREF;         /* MONITOR normal return status */
8114  1
```

```
8115    1       PROMPT = NO;                    /* Indicate no more prompting for subcommands */
8116    1
8117    1       RETURN (NORMAL);                /* Return with status code */
8118    1
8119    1       END EXIT_CMD;
8120
8121
```

```
8122          EXECUTE_CMD: Procedure Returns (Fixed Binary(31));              /* Routine to execute a MONITOR command file */
8123    1
8124    1     /***
8125    1     /*
8126    1     /* FUNCTIONAL DESCRIPTION:
8127    1     /*
8128    1     /*      EXECUTE_CMD
8129    1     /*
8130    1     /*      This is the routine invoked by CLI$DISPATCH when the EXECUTE command is given to MONITOR
8131    1     /*      at the interactive level.  EXECUTE_CMD opens the execute command file, sets the bit flag EXECUTE,
8132    1     /*      and returns to the main interactive loop in MONMAIN (which in turn calls NEXT_EXECUTE_COMMAND).
8133    1     /*      If there is an error opening the file, the error is reported and control returns
8134    1     /*      to the interactive level.
8135    1     /*
8136    1     /* INPUTS:
8137    1     /*
8138    1     /*      Execute filename - this is the file that will be opened so NEXT_EXECUTE_COMMAND can read monitor
8139    1     /*          commands.  The filename is obtained by using CLI$GET_VALUE with the CLD label "EXEC_FILE".
8140    1     /*
8141    1     /* IMPLICIT INPUTS:
8142    1     /*
8143    1     /*      EXECUTE - a bet set to tell the main loop to call NEXT_EXECUTE_COMMAND instead of EXECUTE_COMMAND.
8144    1     /*          NEXET_EXECUTE_COMMAND reads MONITOR commands from the opened file instead of the terminal.
8145    1     /*
8146    1     /* OUTPUTS:
8147    1     /*
8148    1     /*      None.
8149    1     /*
8150    1     /* IMPLICIT OUTPUTS:
8151    1     /*
8152    1     /*      COMMAND_FILE - the file reference associated with the execute command file.  This is also used by
8153    1     /*          NEXT_EXECUTE_COMMAND to read the file.
8154    1     /*
8155    1     /* ROUTINE VALUE:
8156    1     /*
8157    1     /*      MNR$_ERREXECOM  - EXECUTE subcommand cannot be used in an execute command file.
8158    1     /*      MNR$_ERREXEFIL  - CLI$GET_VALUE failed to return the execute file name.
8159    1     /*      MNR$_ERREXEOPN  - Open failure using the execute file name.
8160    1     /*      SS$_NORMAL      - Success.
8161    1     /*
8162    1     /* SIDE EFFECTS:
8163    1     /*
8164    1     /*      None.
8165    1     /*
8166    1     /*/
8167    1
```

```
8168  1        /*
8169  1        /*      !-------------------------------------------------------------------------+
8170  1        /*      !                                                                         !
8171  1        /*      !                              REQUIRE FILE                               !
8172  1        /*      !                                                                         !
8173  1        /*      !-------------------------------------------------------------------------+
8174  1        /*/
8175  1
8176  1        %INCLUDE        MONDEF;                                 /* Monitor utility structure definitions */
8944  1
8945  1        %REPLACE        FILE_SPEC_SIZE          BY 128;         /* Max file spec size */
8946  1        %REPLACE        NOT_SUCCESSFUL          BY '0'B;        /* Failing severity code */
8947  1
8948  1
8949  1        /*
8950  1        /*      !-------------------------------------------------------------------------+
8951  1        /*      !                                                                         !
8952  1        /*      !                    GLOBAL STORAGE DEFINITIONS                           !
8953  1        /*      !                                                                         !
8954  1        /*      !-------------------------------------------------------------------------+
8955  1        /*/
8956  1
8957  1        Declare
8958  1            COMMAND_FILE        FILE RECORD GLOBALDEF;          /* Execute command file */
8959  1
8960  1        /*
8961  1        /*      !-------------------------------------------------------------------------+
8962  1        /*      !                                                                         !
8963  1        /*      !                          EXTERNAL REFERENCES                            !
8964  1        /*      !                                                                         !
8965  1        /*      !-------------------------------------------------------------------------+
8966  1        /*/
8967  1
8968  1        Declare
8969  1            CLI$GET_VALUE       ENTRY(ANY, ANY)                 /* CLE routine to get qualifier values */
8970  1                                RETURNS(FIXED BINARY(31)),
8971  1            MON_ERR             ENTRY (ANY VALUE, ANY, ANY)     /* Routine to log synchronous errors */
8972  1                                OPTIONS(VARIABLE);
8973  1        Declare
8974  1            CURR_ERRCODE        FIXED BINARY(31) GLOBALREF,     /* MONITOR error status code currently expected */
8975  1            EXECUTE             BIT(1) ALIGNED   GLOBALREF,     /* YES => read another command from the execute file
8976  1            NORMAL              FIXED BINARY(31) GLOBALREF;     /* MONITOR normal return status */
8977  1
8978  1        Declare
8979  1            MNR$_ERREXECOM      FIXED BINARY(31) GLOBALREF VALUE,   /* Error message code */
8980  1            MNR$_ERREXEFIL      FIXED BINARY(31) GLOBALREF VALUE,   /* Error message code */
8981  1            MNR$_ERREXEOPN      FIXED BINARY(31) GLOBALREF VALUE;   /* Error message code */
8982  1
8983  1
8984  1        /*
8985  1        /*      !-------------------------------------------------------------------------+
8986  1        /*      !                                                                         !
8987  1        /*      !                             LOCAL STORAGE                               !
8988  1        /*      !                                                                         !
8989  1        /*      !-------------------------------------------------------------------------+
8990  1        /*/
```

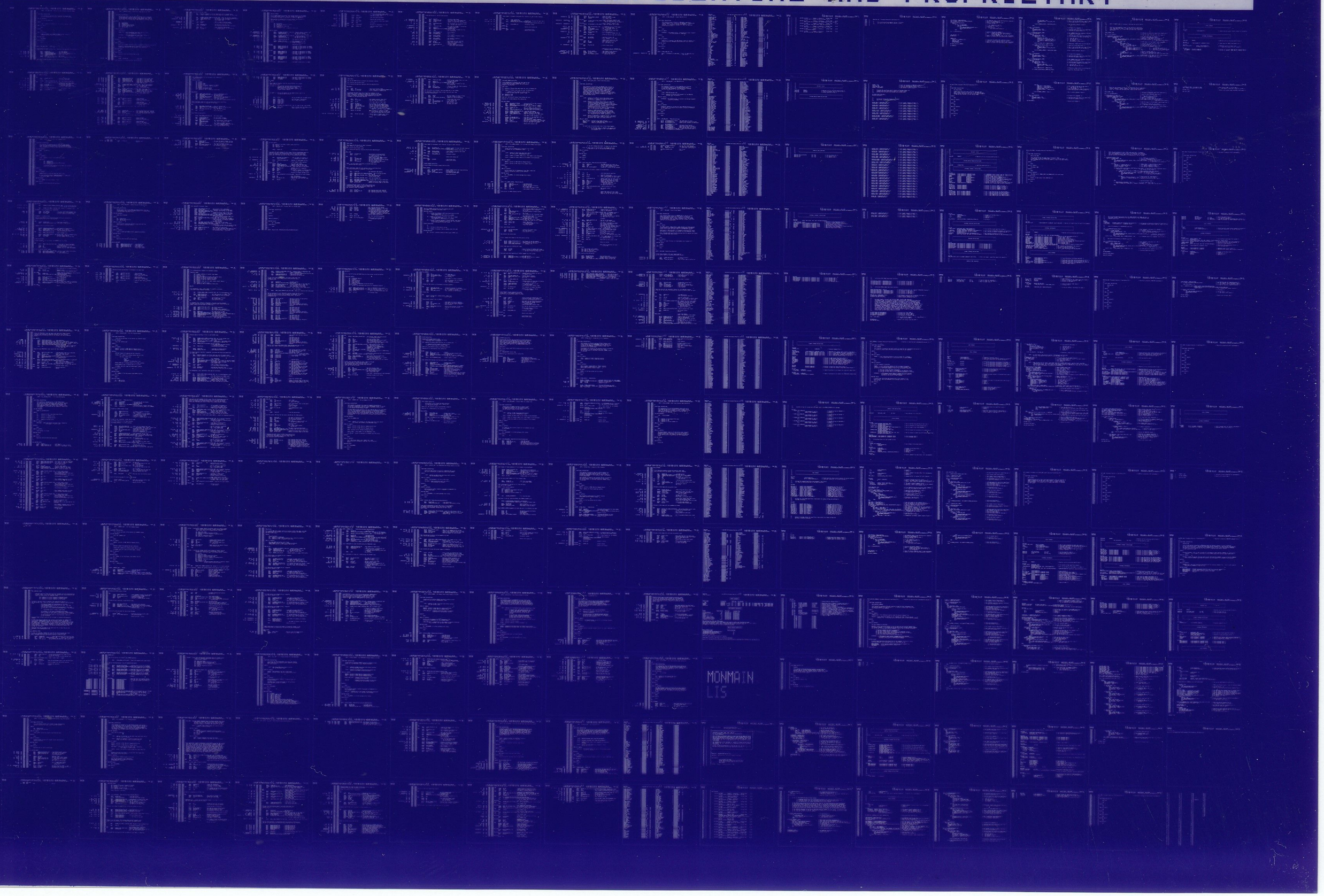
```
8991    1
8992    1       Declare
8993    1           CALL                FIXED BINARY(31),                   /* Holds function value (return status) of called ro
8994    1           STATUS              BIT(1) BASED(ADDR(CALL));           /* Low-order status bit for called routines */
8995    1
8996    1       Declare
8997    1           1 EXEC_FILE_PARM,
8998    1             2 L               FIXED BINARY (31),
8999    1             2 A               POINTER,
9000    1
9001    1           1 EXEC_FILE_VAL,
9002    1             2 L               FIXED BINARY (31),
9003    1             2 A               POINTER,
9004    1
9005    1           EXEC_FILE_NAME      CHAR(9)  STATIC INIT('EXEC_FILE'),
9006    1           EXEC_FILE_STR       CHAR(FILE_SPEC_SIZE) STATIC INIT('MONITOR.MON');
9007    1
9008    1
9009    1       ON UNDEFINEDFILE (COMMAND_FILE) GOTO OPEN_ERROR;            /* Set up the UNDEFINEDFILE condition */
9010    1
9011    1
9012    1       IF EXECUTE = YES                                            /* Make sure there is only 1 level of execute comman
9013    1           THEN DO;
9014    2               CALL MON_ERR(MNR$_ERREXECOM);                       /* Log the error... */
9015    2               RETURN(MNR$_ERREXECOM);                             /* Return to processing the execute command file */
9016    2           END;
9017    1
9018    1       EXECUTE = YES;                                              /* Let everyone know command input is coming from a
9019    1       EXEC_FILE_PARM.L = LENGTH(EXEC_FILE_NAME);                  /* Set the length of the EXEC_FILE descriptor */
9020    1       EXEC_FILE_PARM.A = ADDR(EXEC_FILE_NAME);                    /* Set the address of the EXEC_FILE string */
9021    1       EXEC_FILE_VAL.L =  LENGTH(EXEC_FILE_STR);                   /* Set the length of the execute filename descriptor
9022    1       EXEC_FILE_VAL.A =  ADDR(EXEC_FILE_STR);                     /* Set the address of the execute filename string */
9023    1       CURR_ERRCODE = MNR$_ERREXEFIL;                              /* Set MONITOR code for signaled errors */
9024    1       CALL = CLI$GET_VALUE(EXEC_FILE_PARM,EXEC_FILE_VAL);         /* Get EXECUTE file name */
9025    1       IF STATUS = NOT_SUCCESSFUL                                  /* If failed, */
9026    1           THEN DO;
9027    2               EXECUTE = NO;                                       /* Error, no longer doing an EXECUTE subcommand */
9028    2               CALL MON_ERR(MNR$_ERREXEFIL,CALL);                  /* Log the error ... */
9029    2               RETURN(MNR$_ERREXEFIL);                             /* ... and return with status */
9030    2               END;
9031    1
9032    1       OPEN FILE(COMMAND_FILE) INPUT SEQUENTIAL TITLE(EXEC_FILE_STR)   /* Open the execute command file */
9033    1           ENVIRONMENT(DEFAULT_FILE_NAME('.MON'));                 /*  with .MON for default file type */
9034    1
9035    1       CURR_ERRCODE = 0;                                           /* Set MONITOR code for signaled errors */
9036    1       RETURN (NORMAL);                                            /* Return with success code */
9037    1
9038    1       OPEN_ERROR:
9039    1           EXECUTE = NO;                                           /* Indicate no more from the execute file */
9040    1           CURR_ERRCODE = 0;                                       /* Reset to default MONITOR code in case subcommand
9041    1           CALL MON_ERR(MNR$_ERREXEOPN);                           /* Log the error ... */
9042    1           RETURN(MNR$_ERREXEOPN);                                 /* ... and return with status */
9043    1
9044    1       END EXECUTE_CMD;
```

COMMAND LINE

```
 ------- ----

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

PR
VO



MONMAIN
LIS