```
BBBBBBBBBBBB        AAAAAAAAA     DDDDDDDDDDDD
BBBBBBBBBBBB        AAAAAAAAA     DDDDDDDDDDDD
BBBBBBBBBBBB        AAAAAAAAA     DDDDDDDDDDDD
BBB         BBB  AAA         AAA  DDD         DDD
BBB         BBB  AAA         AAA  DDD         DDD
BBB         BBB  AAA         AAA  DDD         DDD
BBB         BBB  AAA         AAA  DDD         DDD
BBB         BBB  AAA         AAA  DDD         DDD
BBB         BBB  AAA         AAA  DDD         DDD
BBBBBBBBBBBB     AAA         AAA  DDD         DDD
BBBBBBBBBBBB     AAA         AAA  DDD         DDD
BBBBBBBBBBBB     AAA         AAA  DDD         DDD
BBB         BBB  AAAAAAAAAAAAAAA  DDD         DDD
BBB         BBB  AAAAAAAAAAAAAAA  DDD         DDD
BBB         BBB  AAAAAAAAAAAAAAA  DDD         DDD
BBB         BBB  AAA         AAA  DDD         DDD
BBB         BBB  AAA         AAA  DDD         DDD
BBB         BBB  AAA         AAA  DDD         DDD
BBBBBBBBBBBB     AAA         AAA  DDDDDDDDDDDD
BBBBBBBBBBBB     AAA         AAA  DDDDDDDDDDDD
BBBBBBBBBBBB     AAA         AAA  DDDDDDDDDDDD
```

FILEID**BADINIT

```
BBBBBBBB      AAAAAA    DDDDDDDD      IIIIII    NN      NN    IIIIII    TTTTTTTTTT
BBBBBBBB      AAAAAA    DDDDDDDD      IIIIII    NN      NN    IIIIII    TTTTTTTTTT
BB      BB  AA      AA  DD      DD      II      NN      NN      II          TT
BB      BB  AA      AA  DD      DD      II      NN      NN      II          TT
BB      BB  AA      AA  DD      DD      II      NNNN    NN      II          TT
BB      BB  AA      AA  DD      DD      II      NNNN    NN      II          TT
BBBBBBBB    AA      AA  DD      DD      II      NN  NN  NN      II          TT
BBBBBBBB    AA      AA  DD      DD      II      NN  NN  NN      II          TT
BB      BB  AAAAAAAAAA  DD      DD      II      NN    NNNN      II          TT
BB      BB  AAAAAAAAAA  DD      DD      II      NN    NNNN      II          TT
BB      BB  AA      AA  DD      DD      II      NN      NN      II          TT        ....
BB      BB  AA      AA  DD      DD      II      NN      NN      II          TT        ....
BBBBBBBB    AA      AA  DDDDDDDD      IIIIII    NN      NN    IIIIII        TT        ....
BBBBBBBB    AA      AA  DDDDDDDD      IIIIII    NN      NN    IIIIII        TT        ....

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

```
MODULE badinit( %TITLE 'Analyze/Media Initialization Module'
                IDENT = 'V04-000') =
BEGIN
!
!****************************************************************************
!*                                                                          *
!*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                 *
!*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                  *
!*  ALL RIGHTS RESERVED.                                                    *
!*                                                                          *
!*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED   *
!*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE   *
!*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER   *
!*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY   *
!*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY   *
!*  TRANSFERRED.                                                            *
!*                                                                          *
!*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE   *
!*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT   *
!*  CORPORATION.                                                            *
!*                                                                          *
!*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS   *
!*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                 *
!*                                                                          *
!*                                                                          *
!****************************************************************************
!
!
!++
!
! Facility:
!
!       Analyze/Media
!
! Abstract:
!
!       This module contains the procedures used to parse the command string
!       and allocate/initialize the control and storage structures which define
!       the device's analysis context.
!
! Environment:
!
!       VAX/VMS User Mode, Non-Privileged
!
! Author:
!
!       Michael T. Rhodes,              Creation Date: July, 1982
!
! Modified By:
!
!       V03-004 MTR0009         Michael T. Rhodes        3-Jul-1984
!               Fix processing of /SHOW=BEFORE qualifier to obtain
!               the correct SDBSF and generation of listings.
!
!       V03-003 MTR0003         Michael T. Rhodes       11-Feb-1983
!               Add support for Stand Alone BAD.  Routine 'alloc_mem'
!               has been changed to be a GLOBAL routine, renamed to
```

```
1 !                     'bad$alloc_mem'.
1 !
1 !             V03-002 MTR0002         Michael T. Rhodes       20-Jan-1983
1 !                     Use shared message BADVALUE instead of INVALNUM.
1 !
1 !             V03-001 MTR0001         Michael T. Rhodes       15-Dec-1982
1 !                     Add the /LOG qualifier and the keyword PATTERN to the
1 !                     /EXERCISE qualifier.  Several ROUTINEs have been added
1 !                     to support these features:
1 !                         'bad_value', 'get_pattern', and 'parse_value'
1 !
1 !--
1
```

```
1 %SBTTL 'Declarations'
1 !
1 ! Include Files:
1 !
1 REQUIRE 'lib$:baddef';                                          ! Define BAD's structures.
1 LIBRARY 'SYS$LIBRARY:LIB';                                      ! Define VMS structures.
1 LIBRARY 'SYS$LIBRARY:TPAMAC';                                   ! TPARSE macro definitions.
1
1 !
1 ! Table of Contents:
1 !
1 FORWARD ROUTINE
1     bad$alloc_mem         : NOVALUE,                            ! Standard memory getter!
1     bad$init              : NOVALUE,                            ! Process the command line qualifiers and init struc
1     addr_error,                                                 ! Handle address range errors.
1     bad_value,                                                  ! Bad numeric value error routine.
1     count_term,                                                 ! Number of terms in expression.
1     get_mdbsf             : NOVALUE,                            ! Get a valid copy of the MDBSF.
1     get_nlt_sdbsf,                                              ! Get a valid copy of the SDBSF from a non last trac
1     get_pattern           : NOVALUE,                            ! Process user supplied test pattern(s).
1     get_sdbsf             : NOVALUE,                            ! Get a valid copy of the SDBSF.
1     init_sdbsf            : NOVALUE,                            ! Initialize the header of the SDBSF.
1     parse_value           : NOVALUE,                            ! Parse and process bad block info.
1     process_badblocks,                                          ! Process user supplied bad block information.
1     rebuild_dbsfs         : NOVALUE,                            ! Rebuild the MDBSF and SDBSF.
1     recover_dbsfs,                                              ! Attempt to rebuild/recover an SDBSF.
1     syntax_err,                                                 ! Error action routine for TPARSE.
1     val_log_adr,                                                ! Validate the user supllied logical address.
1     val_phy_adr;                                                ! Validate the user supplied physical address.
1
1 !
1 ! Private Storage
1 !
1 BIND                                                            ! The list below contains the
1     after =     $DESCRIPTOR ('AFTER'),                          ! strings used to parse the
1     bad_blocks =$DESCRIPTOR ('BAD_BLOCKS'),                     ! command qualifiers and keywords.
1     before =    $DESCRIPTOR ('BEFORE'),                         ! Also, some of the strings are
1     command_lin=$DESCRIPTOR ('$LINE'),                          ! used in the various messages
1     count =     $DESCRIPTOR ('COUNT'),                          ! issued from the utility.
1     cylinder =  $DESCRIPTOR ('CYLINDER'),
1     device =    $DESCRIPTOR ('DEVICE'),
1     exercise =  $DESCRIPTOR ('EXERCISE'),
1     full =      $DESCRIPTOR ('FULL'),
1     keep =      $DESCRIPTOR ('KEEP'),
1     log =       $DESCRIPTOR ('LOG'),
1     logical_blk=$DESCRIPTOR ('LOGICAL_BLOCK'),
1     mdbsf =     $DESCRIPTOR ('MDBSF'),
1     output =    $DESCRIPTOR ('OUTPUT'),
1     pattern =   $DESCRIPTOR ('PATTERN'),
1     prompt =    $DESCRIPTOR ('BAD_BLOCKS = '),
1     retry =     $DESCRIPTOR ('RETRY'),
1     sdbsf =     $DESCRIPTOR ('SDBSF'),
1     sector =    $DESCRIPTOR ('SECTOR'),
1     show_qual = $DESCRIPTOR ('SHOW'),
1     track =     $DESCRIPTOR ('TRACK');
1
1 !
```

```
! External references:
!
EXTERNAL ROUTINE
    bad$check_ltk         : ADDRESSING_MODE (GENERAL),    ! Check last track device for a user supplied entry.
    bad$check_nlt         : ADDRESSING_MODE (GENERAL),    ! Check non last track device for a user supplied en
    bad$cvt_ltk_long      : ADDRESSING_MODE (GENERAL),    ! Convert last track format to longwords.
    bad$cvt_nlt_long      : ADDRESSING_MODE (GENERAL),    ! Convert non last track format to longwords.
    bad$cvt_phy_log       : ADDRESSING_MODE (GENERAL),    ! Convert physical address to logical.
    bad$prepare_report    : ADDRESSING_MODE (GENERAL),    ! Open the output file and generate report heading.
    bad$produce_report    : ADDRESSING_MODE (GENERAL),    ! Produce a listing of the bad block file(s).
    bad$sta_init          : WEAK,                         ! Perform initialization for execution in the Stand
    bad$str_trim          : WEAK,                         ! Trim trailing blanks, tabs, and carriage return fr
    bad$sync_io           : ADDRESSING_MODE (GENERAL),    ! Perform synchronous IO.
    bad$validate_pack     : WEAK,                         ! Set Volume Valid bit in the UCB when executing Sta
    CHECKSUM2             : ADDRESSING_MODE (GENERAL),    ! Compute block checksum.
    CLI$GET_VALUE         : ADDRESSING_MODE (GENERAL),    ! CLI call back routine.
    CLI$PRESENT           : ADDRESSING_MODE (GENERAL),    ! CLI call back routine.
    LIB$GET_INPUT         : ADDRESSING_MODE (GENERAL),    ! Library input routine.
    LIB$TPARSE            : ADDRESSING_MODE (GENERAL);    ! Library State Table Parser.

EXTERNAL
    bad$gl_bad_term       : VECTOR [4, LONG],             ! Bad block information vector.
    bad$gb_block_fact     : BYTE,                         ! Number of sectors per block.
    bad$gl_bytes_cyl,                                     ! Total number of bytes per cylinder.
    bad$gl_bytes_trk,                                     ! Total number of bytes per track.
    bad$gl_chan,                                          ! Channel number for device.
    bad$ga_comnd_line     : $BBLOCK [dsc$c_s_bln],        ! Address of command line buffer descriptor.
    bad$gl_context        : BITVECTOR [32],               ! Global control context flags.
    bad$gl_cylinders,                                     ! Number of cylinders on medium.
    bad$ga_bufadr         : VECTOR [2, LONG],             ! Address vector of the data transfer buffers.
    bad$gl_devchar        : $BBLOCK,                      ! Device characteristics.
    bad$gl_devclass,                                      ! Class of the device.
    bad$ga_device         : $BBLOCK [dsc$c_s_bln],        ! Address of device name descriptor.
    bad$ga_devnam         : $BBLOCK [64],                 ! Device name buffer.
    bad$gl_devnam,                                        ! Size of device name.
    bad$gq_devnam         : $BBLOCK [dsc$c_s_bln],        ! Device name descriptor.
    bad$ga_filespec       : $BBLOCK [dsc$c_s_bln],        ! Address of filespec buffer descriptor.
    bad$gl_func,                                          ! IO function code for $QIO.
    bad$ga_getdvi         : $BBLOCK,                      ! Address of the $GETDVI item list.
    bad$ga_input_desc     : $BBLOCK [dsc$c_s_bln],        ! Generic input descriptor.
    bad$gq_iosb           : VECTOR [4, LONG],             ! IO status blocks.
    bad$gl_maxblock,                                      ! Highest user addressable LBN on device.
    bad$ga_mdbsf          : REF $BBLOCK,                  ! Address of the Manufacturers Detected Bad Sector F
    bad$gl_mdbsf_ptr,                                     ! Disk address of the MDBSF.
    bad$gl_pagcnt,                                        ! Number of 512 byte pages for 1 transfer.
    bad$ga_sdbsf          : REF $BBLOCK,                  ! Address of the Software Detected Bad Sector File.
    bad$gl_sdbsf_ptr,                                     ! Pointer to the first available entry in the SDBSF.
    bad$gl_sectors,                                       ! Number of sectors per track.
    bad$gl_sector_siz,                                    ! Size of the physical sector on a drive.
    bad$gl_serialnum,                                     ! Pack serial number.
    bad$gl_status,                                        ! Global status.
    bad$gb_term_count     : BYTE,                         ! Number of terms used to describe bad block.
    bad$ga_tpb,                                           ! Address of the Test Pattern Buffer.
    bad$gl_tracks,                                        ! Number of tracks per cylinder.
    bad$gl_trnsfr_cnt;                                    ! Number of bytes per transfer.

!
```

```
! Define message codes...
!
EXTERNAL LITERAL
    bad$_aligndisk,                         ! The cartridge is an alignment disk.
    bad$_assign,                            ! Failed to assign a channel.
    bad$_badvalue,                          ! Invalid keyword value.
    bad$_devnotblk,                         ! The device is not block structured.
    bad$_devnotfor,                         ! The device is not mounted foreign.
    bad$_devrct,                            ! Device has revector caching enabled.
    bad$_getdvi,                            ! $GETDVI failed.
    bad$_insvirmem,                         ! Insufficient virtual memory.
    bad$_invalqual,                         ! Invalid qualifier.
    bad$_ivblknum,                          ! Invalid block number.
    bad$_ivblkent,                          ! Misiing or invalid block entity.
    bad$_lststring,                         ! Used to list an ascid string.
    bad$_mdbsfcrupt,                        ! MDBSF is corrupt.
    bad$_mdbsfrfail,                        ! Failed to read MDBSF.
    bad$_nobadinfo,                         ! No bad block info.
    bad$_nooutqual,                         ! No /OUTPUT qualifier.
    bad$_rebldwarn,                         ! Failed to write enough new MDBSF and SDBSF files.
    bad$_sdbsffull,                         ! SDBSF is full.
    bad$_sdbsfrfail;                        ! Failed to read the SDBSF.

!
! Macros:
!
MACRO
    def_tparse_args =
        BUILTIN AP;
        BIND tparse_args = AP : REF $BBLOCK;
        %;
```

```
%SBTTL  'State table definitions'
!++
!
! Functional Description:
!
!       The state table provides the definition of legal grammar for the
!       acceptance of bad block information.  The legal forms of bad block
!       information are:
!
!               1.  lbn
!
!               2.  lbn:count
!
!               3.  sector.track.cylinder
!
!               4.  sector.track.cylinder:count
!
!       Each of the elements may be specified in any of the 3 valid radices,
!       OCTAL, DECIMAL, or HEXADECIMAL.  Note that ANY of the elements may be
!       of ANY radix, no restriction is imposed on the user requiring that the
!       same radix be used for all terms.  The default radix is DECIMAL.
!
!       A simple action routine "count_term" is called by TPARSE on our
!       behalf.  It increments a count of the number of terms processed and
!       stores the values taken from the TPARSE control block at the location
!       indicated by "bad$gl_bad_term [.bad$gb_term_count]".  The count will
!       later be used to convert the bad block information into either logical
!       block number or in physical address form (plus a count value if included).
!
!       An additional state table is also defined, which provides a simple
!       parsing mechanism for numeric values.  This is used by for parsing the
!       keyword PATTERN's value(s).
!
!--


$INIT_STATE (state_tbl, key_tbl);

$STATE  (,
        ((number),, count_term),
        ((syntax)));

$STATE  (,
        ('.', track_number),
        (TPA$_LAMBDA)
        );

$STATE  (colon,
        (':', blkcnt),
        (TPA$_LAMBDA)
        );

$STATE  (blkend,
        (TPA$_EOS, TPA$_EXIT),
        ((syntax)));

$STATE  (blkcnt,
```

```
                ((number), blkend, count_term),
                ((syntax)));

$STATE  (track_number,
        ((number), count_term),
        ((syntax)));

$STATE  (,
        ('.', cylinder_number),
        ((syntax)));

$STATE  (cylinder_number,
        ((number), colon, count_term),
        ((syntax)));
!
! The following portion of the state table is shared to allow the parsing
! of integer numbers of various radices.
!
$INIT_STATE (number_tbl, numkey_tbl);

$STATE  (,
        ((number), count_term),
        ((bad_val)));

$STATE  (number,
        (TPA$_DECIMAL, TPA$_EXIT),
        ('%'));

$STATE  (,
        ('d', decnum),
        ('D', decnum),
        ('x', hexnum),
        ('X', hexnum),
        ('o', octnum),
        ('O', octnum));

$STATE  (octnum,
        (TPA$_OCTAL, TPA$_EXIT));

$STATE  (decnum,
        (TPA$_DECIMAL, TPA$_EXIT));

$STATE  (hexnum,
        (TPA$_HEX, TPA$_EXIT));

$STATE  (bad_val,
        (TPA$_SYMBOL,,  bad_value),
        (TPA$_STRING,,  bad_value),
        (TPA$_LAMBDA,,  bad_value),
        (TPA$_ANY,,     bad_value),
        (TPA$_EOS,,     bad_value)
        );

$STATE  (syntax,
        (TPA$_SYMBOL,,  syntax_err),
        (TPA$_STRING,,  syntax_err),
        (TPA$_LAMBDA,,  syntax_err),
```

```
;  333      P 0450  1        (TPA$_ANY,,       syntax_err),
;  334      P 0451  1        (TPA$_EOS,,       syntax_err)
;  335        0452  1        );
;  336        0453  1
```

```
1 %SBTTL 'bad$init -- Initialization Procedure'
1
1 GLOBAL ROUTINE bad$init : NOVALUE =
1 !++
1 !
1 ! Functional Description:
1 !
1 !     This procedure processes the qualifiers specified/defaulted in the command
1 !     line.  The device is validated and an IO channel established.  If an output
1 !     file is requested, it is opened at this time.
1 !
1 ! Calling Sequence:
1 !
1 !     bad$init ();                              ! Parse the command line and
1 !                                               ! process/initialize structures.
1 !
1 ! Implicit Inputs:
1 !
1 !     Pointers to control and data structures.
1 !
1 ! Implicit Outputs:
1 !
1 !     All of the global control and data structures have been allocated and
1 !     initialized.
1 !
1 !--
2 BEGIN
2
2 bad$gl_context [ctx_v_init] = TRUE;                        ! Flag initialization phase.
2
2 !
2 ! Initialize descriptors.
2 !
2 CH$FILL (0, dsc$c_s_bln, bad$ga_comnd_line);
2 CH$FILL (0, dsc$c_s_bln, bad$ga_device);
2 CH$FILL (0, dsc$c_s_bln, bad$ga_input_desc);
2 CH$FILL (0, dsc$c_s_bln, bad$ga_filespec);
2 bad$ga_comnd_line [dsc$b_class] = dsc$k_class_d;           ! Command line descriptor.
2 bad$ga_device [dsc$b_class] = dsc$k_class_d;               ! Device name descriptor.
2 bad$ga_input_desc [dsc$b_class] = dsc$k_class_d;           ! Input line descriptor.
2 bad$ga_filespec [dsc$b_class] = dsc$k_class_d;             ! Output file spec descriptor.
2
2 IF bad$sta_init NEQ 0                                       ! Are we executing Stand Alone or under VMS?
2 THEN    bad$sta_init ();                                    ! Stand Alone -- perform required initialization.
2
2 !
2 ! Save the command line and get the device spec.
2 !
2 CLI$GET_VALUE (command_lin, bad$ga_comnd_line);            ! Get a local copy of the command line.
2 IF bad$str_trim NEQ 0                                       ! Trim the command line if we are stand alone.
2 THEN    bad$str_trim (bad$ga_comnd_line);
2
2 CLI$GET_VALUE (device, bad$ga_device);                     ! Copy the device name.
2 IF bad$str_trim NEQ 0                                       ! Trim the device specification if we are stand alon
2 THEN    bad$str_trim (bad$ga_device);
2
3 IF NOT (bad$gl_status =
```

```
P  3         $GETDVIW (DEVNAM = bad$ga_device,                    ! Get the device specifics.
   3                   ITMLST = bad$ga_getdvi))
   2  THEN                                                        ! If not obtainable -- STOP!
   2      SIGNAL_STOP (bad$_getdvi, 1, bad$ga_device, .bad$gl_status);
   2
   2  IF .bad$gl_devclass NEQ dc$_disk                            ! Is this a valid device type?
   2  THEN                                                        ! If not -- STOP!
   2      SIGNAL_STOP (bad$_devnotblk, 1, bad$ga_device);
   2
   2  IF .bad$gl_devchar [dev$v_rct]                              ! Revector cached device?
   2  THEN                                                        ! If so, STOP, don't waste time
   2      SIGNAL_STOP (bad$_devrct, 1, bad$ga_device);            ! essentially doing a NOP!
   2
   2  IF NOT .bad$gl_devchar [dev$v_for]                          ! Device must be mounted /FOREIGN
   2  AND bad$sta_init EQL 0                                      ! (while executing under VMS),
   2  THEN                                                        ! because we do not have enough
   2      SIGNAL_STOP (bad$_devnotfor, 1, bad$ga_device);         ! access privilege otherwise.
   2
   3  IF NOT (bad$gl_status =
P  3         $ASSIGN (DEVNAM = bad$ga_device,                     ! If no IO channel, we must STOP!
   3                  CHAN = bad$gl_chan))
   2  THEN
   2      SIGNAL_STOP (bad$_assign, 1, bad$ga_device, .bad$gl_status);
   2
   2  IF bad$validate_pack NEQ 0                                  ! Are we executing Stand Alone?
   2  THEN    bad$validate_pack ();                               ! Yes, set the Volume Valid bit in the UCB.
   2
   2  bad$gq_devnam [dsc$w_length] = .bad$gl_devnam;              ! Build the device name descriptor.
   2  bad$gq_devnam [dsc$b_dtype] = dsc$k_dtype_t;                ! Data type is text.
   2  bad$gq_devnam [dsc$b_class] = dsc$k_class_d;                ! Dynamic descriptor.
   2  bad$gq_devnam [dsc$a_pointer] = bad$ga_devnam;              ! Buffer address.
   2
   2  IF .bad$gl_maxblock GTR bad$k_nltmaxblk                     ! Is this a last track device?
   2  THEN
   3      BEGIN
   3      bad$gl_context [ctx_v_ltdevice] = TRUE;                 ! Default the device type to "last track"
   4      bad$gb_block_fact = (.bad$gl_tracks *                   ! Compute blocking factor.
   4                           .bad$gl_cylinders *
   3                           .bad$gl_sectors) /
   3                           .bad$gl_maxblock;
   3      bad$gl_mdbsf_ptr = .bad$gl_maxblock -                   ! Set beginning address of last track.
   3                         .bad$gl_sectors /
   3                         .bad$gb_block_fact;
   3      bad$gl_sector_siz = bad$k_page_size / .bad$gb_block_fact;    ! Compute the sector size (in bytes).
   3      bad$gl_bytes_trk = .bad$gl_sectors * .bad$gl_sector_siz;     ! Compute number of bytes per track.
   3      bad$gl_bytes_cyl = .bad$gl_bytes_trk * .bad$gl_tracks;       ! Compute number of bytes per cylinder.
   3      bad$gl_trnsfr_cnt = MINU (.bad$gl_bytes_cyl,                 ! Select largest transfer value.
   3                                bad$k_max_xfer_cnt);
   3      END
   2  ELSE
   2      bad$gl_trnsfr_cnt = bad$k_max_xfer_cnt;                 ! Non last track device, make large transfers.
   2
   2  bad$gl_pagcnt = .bad$gl_trnsfr_cnt / bad$k_page_size;       ! Compute memory allocation size in pages.
```

```
2 %SBTTL    'Process qualifiers'
2 !++
2 !
2 ! Process qualifiers.
2 !
2 !     This portion of the procedure sets/clears the various bits in the
2 !     context bits according to the presence of the various qualifiers
2 !     and keywords.
2 !
2 !     The actual processing of the qualifiers/keywords will be performed
2 !     selectively below.
2 !
2 !--
2 bad$gl_context [ctx_v_badblocks] =      CLI$PRESENT (bad_blocks);       ! /BAD_BLOCKS
2 bad$gl_context [ctx_v_exercise] =       CLI$PRESENT (exercise);         ! /EXERCISE
2 bad$gl_context [ctx_v_keep] =           CLI$PRESENT (keep);             ! /[NO]EXERCISE=KEEP
2 bad$gl_context [ctx_v_log] =            CLI$PRESENT (log);              ! /LOG
2 bad$gl_context [ctx_v_output] =         CLI$PRESENT (output);           ! /OUTPUT
2 bad$gl_context [ctx_v_retry] =          CLI$PRESENT (retry);            ! /RETRY
2 bad$gl_context [ctx_v_show] =           CLI$PRESENT (show_qual);        ! /SHOW
2
2 !
2 ! END of generic qualifier/keyword processing...
2 !
2
2 !
2 !     If the device is a last track device, then allocate the MDBSF and SDBSF
2 !     buffers and attempt to obtain the MDBSF.  If the device is a non last
2 !     track device, simply allocate the SDBSF buffer.
2 !
2 !     (The SDBSF is only preserved if so directed by the KEEP keyword)
2 !
2 bad$gl_func = IO$_READLBLK;                                 ! Set default IO function code.
2 IF .bad$gl_context [ctx_v_ltdevice]
2 THEN
3     BEGIN
3     bad$alloc_mem (bad$k_page_size * 2, bad$ga_mdbsf);  ! Allocate the detected bad sector file buffers.
3     bad$ga_sdbsf = .bad$ga_mdbsf + bad$k_page_size;     ! Seperate the buffers.
3     get_mdbsf ();                                       ! Obtain a valid copy of the MDBSF.
3     END
2 ELSE
2     bad$alloc_mem (bad$k_page_size, bad$ga_sdbsf);      ! Allocate a single buffer.
2
2 !
2 ! *                                                     *
2 !*** Process individual qualifiers for specific actions. ***
2 ! *                                                     *
2 !
2 SELECT TRUE OF SET
2
```

```
2 %SBTTL    'Process /OUTPUT qualifier'
2 !++
2 !
2 ! Process /OUTPUT qualifier.
2 !
2 !     If requested get the file specified (or defaulted) by the user and
2 !     the output file.  The default specification of /SHOW=AFTER is also
2 !     set (just in case it wasn't specified in the command line).
2 !
2 !     The default is /NOOUTPUT.
2 !
2 !--
2 [.bad$gl_context [ctx_v_output]]:
3       BEGIN
3       CLI$GET_VALUE (output, bad$ga_filespec);              ! Get the filespec if supplied.
3       bad$prepare_report ();                                ! Open the file.
3       bad$gl_context [ctx_v_show_after] = TRUE;             ! Default to /SHOW=AFTER if it is not supplied.
3
2       END;     ! of /OUTPUT processing
2
```

```
%SBTTL  'Process /SHOW qualifier'
!++
!
! Process /SHOW qualifier.
!
!     This procedure processes the /SHOW qualifier which will produce the
!     following actions:
!
!          1.  If /SHOW=BEFORE is specified the listing is produced from the
!              the current bad block file(s) as they exist in our buffers.
!              The AFTER qualifier value context switch will be toggled by a
!              call to CLI$PRESENT.
!
!          2.  If /SHOW or /SHOW=AFTER is specified the the context bit
!              "show_after" will be set, resulting in the contents of the bad
!              block file(s) being listed at the conclusion of processing.
!
!--
[.bad$gl_context [ctx_v_show]]:
    BEGIN                                                   ! If /SHOW is specified without /OUTPUT,
    IF NOT .bad$gl_context [ctx_v_output]                   ! then -- STOP!  We do not have a valid
    THEN                                                    ! output file to write the report to!
        SIGNAL_STOP (bad$_nooutqual, 0, bad$_lststring, 1, bad$ga_comnd_line);

    IF (bad$gl_context [ctx_v_show_before] = CLI$PRESENT (before))
    THEN                                                    ! Should we produce a report before
        BEGIN                                               ! testing the media?  If so, then
        IF .bad$gl_context [ctx_v_ltdevice]                 ! obtain a copy of the old SDBSF,
        THEN    get_sdbsf ()                                ! produce the report, and toggle the
        ELSE    bad$gl_context [ctx_v_nlt_sdbsf] = get_nlt_sdbsf (); ! show_after context flag.
        IF .bad$gl_context [ctx_v_ltdevice] OR .bad$gl_context [ctx_v_nlt_sdbsf]
        THEN    bad$produce_report ();
        bad$gl_context [ctx_v_show_after] = CLI$PRESENT (after);
        END
    ELSE
        bad$gl_context [ctx_v_show_after] = TRUE;           ! Otherwise the default is show_after.

    END;    ! of /SHOW processing
```

```
2 %SBTTL  'Process the KEEP keyword'
2 !++
2 !
2 ! Process the KEEP keyword ** ALWAYS PERFORMED **.
2 !
2 !     This portion of the procedure will determine whether the contents of the
2 !     old SDBSF should be preserved.
2 !
2 !     The default is dependent on the EXERCISE context.  If the context is
2 !     to EXERCISE the device, the default is NOT to preserve the old SDBSF
2 !     contents.  This allows the user to purge the allocation of user supplied
2 !     bad blocks from the SDBSF.  If the context is NOEXERCISE, then KEEP is
2 !     implicit (we must protect and preserve the contents of the SDBSF, since
2 !     it would be too easy to blow away the SDBSF in this mode), resulting in a
2 !     potentially useless piece of media (at least until it was analyzed again).
2 !
2 !--
2 [ALWAYS]:
2
3     BEGIN
3     IF .bad$gl_context [ctx_v_keep] OR                    ! Should we preserve the old SDBSF?
4     (NOT .bad$gl_context [ctx_v_exercise] AND NOT .bad$gl_context [ctx_v_show_before])
3     THEN
3         IF .bad$gl_context [ctx_v_ltdevice]
3         THEN    get_sdbsf ()                              ! Yes, obtain a valid copy of the SDBSF.
3         ELSE    get_nlt_sdbsf ()                          ! Yes, obtain a valid copy of the non last track SDB
3     ELSE
3         init_sdbsf ();                                    ! No, create a new copy of the SDBSF.
3
2     END;    ! of KEEP keyword processing
2
```

```
2 %SBTTL  'Process the /BAD_BLOCKS qualifier'
2 !++
2 !
2 ! Process the /BAD_BLOCKS qualifier.
2 !
2 !     This qualifier may have an associated set of bad block information.
2 !     Any combination of the valid bad block information may be specified
2 !     (in a comma seperated list, utilizing any combination of radix directives).
2 !     The default radix is DECIMAL.
2 !
2 !     Bad block information formats:
2 !
2 !         none                    - If no input is specified with the /BAD_BLOCKS
2 !                                   qualifier, the utility will prompt the user
2 !                                   for the required information.
2 !
2 !         lbn                     - Allocate a single block to the bad block file.
2 !
2 !         lbn:count               - Allocate a range of contiguous blocks starting
2 !                                   at lbn to the bad block file.
2 !
2 !         sec.trk.cyl             - Allocate a single physical sector to the bad
2 !                                   block file.
2 !
2 !         sec.trk.cyl:count       - Allocate a range of contigous physical sectors
2 !                                   to the bad block file.
2 !
2 !     Depending on whether this is a last track or non last track device the bad
2 !     block information will be converted to the required format and entered into
2 !     the SDBSF, if it does not already exist in either the MDBSF or the SDBSF.
2 !
2 !--
2 [.bad$gl_context [ctx_v_badblocks]]:
3     BEGIN
3     LOCAL
3         value_present;
3
3     value_present = FALSE;
3     WHILE bad$gl_status = CLI$GET_VALUE (bad_blocks, bad$ga_input_desc) DO
4         BEGIN                                         ! Bad blocks have been specified
4         value_present = TRUE;                         ! in the command line.
4         process_badblocks ();                         ! Process the current bad block set.
3         END;
3
3     IF NOT .value_present                             ! Should we prompt for the bad blocks?
3     THEN                                              ! Yes.
4         BEGIN
4         bad$gl_context [ctx_v_interactive] = TRUE;    ! Allow duplicate block messages to be displayed.
4         WHILE bad$gl_status = LIB$GET_INPUT (bad$ga_input_desc, prompt)
4         DO
5             BEGIN                                     ! Read a bad block set from the user.
5             IF .bad$gl_status EQL RMS$_EOF            ! Has he terminated the input?
5             THEN EXITLOOP;                            ! Yes -- move on to the next phase.
5             process_badblocks ();                     ! No, process the bad block set.
4             END;
3         END;
3     bad$gl_context [ctx_v_interactive] = FALSE;       ! Disallow duplicate block messages beyond here.
```

```
    END;  ! of /BAD_BLOCKS processing
```

```
2 %SBTTL    'Process /EXERCISE qualifier'
2 !++
2 !
2 ! Process /EXERCISE qualifier.
2 !
2 !     If the device is supposed to be analyzed, then allocate the data
2 !     and test pattern buffers.
2 !
2 !     Check to see if the user requested a completion log of the number of
2 !     bad blocks detected.
2 !
2 !     If the user supplied a test pattern, then obtain a copy of it (upto
2 !     an OCTAWORD (128 bits) of pattern), which will be stored in the vector
2 !     'bad$gl_bad_term'.
2 !
2 !         ***********************************************************************
2 !         ***                             N O T E                             ***
2 !         *** This MUST be the last operation performed on the vector before ***
2 !         *** returning to the driver, since the routine 'bad$init_buffers'  ***
2 !         *** expects the vector to contain the user supplied test pattern!  ***
2 !         ***********************************************************************
2 !
2 !     The default is /NOEXERCISE.
2 !
2 !--
2
2 [.bad$gl_context [ctx_v_exercise]]:
3     BEGIN
3     bad$gl_context [ctx_v_exercise_full] = CLI$PRESENT (full);  ! Set/Clear the FULL context bit.
3     bad$gl_context [ctx_v_nlt_sdbsf] = true;                    ! A new/updated SDBSF will be written to the
3
4     IF (bad$gl_context [ctx_v_pattern] = CLI$PRESENT (pattern)) ! Set/Clear the PATTERN context bit.
3     THEN
3         get_pattern ();                                         ! Process the user supplied test pattern(s).
3
3     bad$alloc_mem (.bad$gl_pagcnt * bad$k_page_size * 2,
3                 bad$ga_bufadr [0]);                             ! Allocate the data buffers.
3
3     bad$ga_bufadr [1] = .bad$ga_bufadr [0] +                    ! Seperate them into distinct entities.
3                         .bad$gl_pagcnt *
3                         bad$k_page_size;
3
3     bad$alloc_mem (bad$k_page_size, bad$ga_tpb);                ! Allocate the Test Pattern Buffer.
3
2     END;  !of /EXERCISE processing
2
2 TES;     ! of Specific Qualifier processing.
2
2 bad$gl_context [ctx_v_init] = FALSE;                    ! Flag end of initialization phase.
2
1 END;     ! Of GLOBAL ROUTINE bad$init

                                        .TITLE  BADINIT Analyze/Media Initialization Module
                                        .IDENT  \V04-000\

                                        .PSECT  _LIB$STATE$,NOWRT, SHR, PIC,1
```

```
           00000 STATE_TBL::
                               .BLKB   0
      89F8 00000 ;TPA$TYPE
                 U.2:          .WORD   -30216                          ;
     0000* 00002 ;TPA$SUBEXP
                 U.4:          .WORD   <<U.3-U.4>-2>                   ;
 00000000V 00004 ;TPA$ACTION
                 U.5:          .LONG   <<COUNT_TERM-U.5>-4>            ;
      0DF8 00008 ;TPA$TYPE
                 U.6:          .WORD   3576                            ;
     0000* 0000A ;TPA$SUBEXP
                 U.8:          .WORD   <<U.7-U.8>-2>                   ;
      102E 0000C ;TPA$TYPE
                 U.9:          .WORD   4142                            ;
     0000* 0000E ;TPA$TARGET
                 U.11:         .WORD   <<U.10-U.11>-2>                 ;
      05F6 00010 ;TPA$TYPE
                 U.12:         .WORD   1526                            ;
           00012 COLON:        .BLKB   0
      103A 00012 ;TPA$TYPE
                 U.13:         .WORD   4154                            ;
     0000* 00014 ;TPA$TARGET
                 U.15:         .WORD   <<U.14-U.15>-2>                 ;
      05F6 00016 ;TPA$TYPE
                 U.16:         .WORD   1526                            ;
           00018 BLKEND:       .BLKB   0
      11F7 00018 ;TPA$TYPE
                 U.17:         .WORD   4599                            ;
      FFFF 0001A ;TPA$TARGET
                 U.18:         .WORD   -1                              ;
      0DF8 0001C ;TPA$TYPE
                 U.19:         .WORD   3576                            ;
     0000* 0001E ;TPA$SUBEXP
                 U.20:         .WORD   <<U.7-U.20>-2>                  ;
           00020 ;BLKCNT
                 U.14:         .BLKB   0
      99F8 00020 ;TPA$TYPE
                 U.21:         .WORD   -26120                          ;
     0000* 00022 ;TPA$SUBEXP
                 U.22:         .WORD   <<U.3-U.22>-2>                  ;
 00000000V 00024 ;TPA$ACTION
                 U.23:         .LONG   <<COUNT_TERM-U.23>-4>           ;
     0000* 00028 ;TPA$TARGET
                 U.24:         .WORD   <<BLKEND-U.24>-2>               ;
      0DF8 0002A ;TPA$TYPE
                 U.25:         .WORD   3576                            ;
     0000* 0002C ;TPA$SUBEXP
                 U.26:         .WORD   <<U.7-U.26>-2>                  ;
           0002E ;TRACK_NUMBER
                 U.10:         .BLKB   0
      89F8 0002E ;TPA$TYPE
                 U.27:         .WORD   -30216                          ;
     0000* 00030 ;TPA$SUBEXP
                 U.28:         .WORD   <<U.3-U.28>-2>                  ;
 00000000V 00032 ;TPA$ACTION
                 U.29:         .LONG   <<COUNT_TERM-U.29>-4>           ;
```

```
     0DF8  00036 ;TPA$TYPE
                 U.30:   .WORD   3576                          ;
     0000* 00038 ;TPA$SUBEXP
                 U.31:   .WORD   <<U.7-U.31>-2>                ;
     102E  0003A ;TPA$TYPE
                 U.32:   .WORD   4142                          ;
     0000* 0003C ;TPA$TARGET
                 U.34:   .WORD   <<U.33-U.34>-2>               ;
     0DF8  0003E ;TPA$TYPE
                 U.35:   .WORD   3576                          ;
     0000* 00040 ;TPA$SUBEXP
                 U.36:   .WORD   <<U.7-U.36>-2>                ;
           00042 ;CYLINDER_NUMBER
                 U.33:   .BLKB   0
     99F8  00042 ;TPA$TYPE
                 U.37:   .WORD   -26120                        ;
     0000* 00044 ;TPA$SUBEXP
                 U.38:   .WORD   <<U.3-U.38>-2>                ;
 00000000V 00046 ;TPA$ACTION
                 U.39:   .LONG   <<COUNT_TERM-U.39>-4>         ;
     0000* 0004A ;TPA$TARGET
                 U.40:   .WORD   <<COLON-U.40>-2>              ;
     0DF8  0004C ;TPA$TYPE
                 U.41:   .WORD   3576                          ;
     0000* 0004E ;TPA$SUBEXP
                 U.42:   .WORD   <<U.7-U.42>-2>                ;
           00050 NUMBER_TBL::
                         .BLKB   0
     89F8  00050 ;TPA$TYPE
                 U.44:   .WORD   -30216                        ;
     0000* 00052 ;TPA$SUBEXP
                 U.45:   .WORD   <<U.3-U.45>-2>                ;
 00000000V 00054 ;TPA$ACTION
                 U.46:   .LONG   <<COUNT_TERM-U.46>-4>         ;
     0DF8  00058 ;TPA$TYPE
                 U.47:   .WORD   3576                          ;
     0000* 0005A ;TPA$SUBEXP
                 U.49:   .WORD   <<U.48-U.49>-2>               ;
           0005C ;NUMBER
                 U.3:    .BLKB   0
     11F3  0005C ;TPA$TYPE
                 U.50:   .WORD   4595                          ;
     FFFF  0005E ;TPA$TARGET
                 U.51:   .WORD   -1                            ;
     0425  00060 ;TPA$TYPE
                 U.52:   .WORD   1061                          ;
     1064  00062 ;TPA$TYPE
                 U.53:   .WORD   4196                          ;
     0000* 00064 ;TPA$TARGET
                 U.55:   .WORD   <<U.54-U.55>-2>               ;
     1044  00066 ;TPA$TYPE
                 U.56:   .WORD   4164                          ;
     0000* 00068 ;TPA$TARGET
                 U.57:   .WORD   <<U.54-U.57>-2>               ;
     1078  0006A ;TPA$TYPE
                 U.58:   .WORD   4216                          ;
     0000* 0006C ;TPA$TARGET
```

```
                    U.60:     .WORD    <<U.59-U.60>-2>              ;
     1058  0006E  ;TPA$TYPE
                    U.61:     .WORD    4184                         ;
    0000*  00070  ;TPA$TARGET
                    U.62:     .WORD    <<U.59-U.62>-2>              ;
     106F  00072  ;TPA$TYPE
                    U.63:     .WORD    4207                         ;
    0000*  00074  ;TPA$TARGET
                    U.65:     .WORD    <<U.64-U.65>-2>              ;
     144F  00076  ;TPA$TYPE
                    U.66:     .WORD    5199                         ;
    0000*  00078  ;TPA$TARGET
                    U.67:     .WORD    <<U.64-U.67>-2>              ;
           0007A  ;OCTNUM
                    U.64:     .BLKB    0
     15F4  0007A  ;TPA$TYPE
                    U.68:     .WORD    5620                         ;
     FFFF  0007C  ;TPA$TARGET
                    U.69:     .WORD    -1                           ;
           0007E  ;DECNUM
                    U.54:     .BLKB    0
     15F3  0007E  ;TPA$TYPE
                    U.70:     .WORD    5619                         ;
     FFFF  00080  ;TPA$TARGET
                    U.71:     .WORD    -1                           ;
           00082  ;HEXNUM
                    U.59:     .BLKB    0
     15F5  00082  ;TPA$TYPE
                    U.72:     .WORD    5621                         ;
     FFFF  00084  ;TPA$TARGET
                    U.73:     .WORD    -1                           ;
           00086  ;BAD_VAL
                    U.48:     .BLKB    0
     81F1  00086  ;TPA$TYPE
                    U.74:     .WORD    -32271                       ;
00000000V  00088  ;TPA$ACTION
                    U.75:     .LONG    <<BAD_VALUE-U.75>-4>         ;
     81F0  0008C  ;TPA$TYPE
                    U.76:     .WORD    -32272                       ;
00000000V  0008E  ;TPA$ACTION
                    U.77:     .LONG    <<BAD_VALUE-U.77>-4>         ;
     81F6  00092  ;TPA$TYPE
                    U.78:     .WORD    -32266                       ;
00000000V  00094  ;TPA$ACTION
                    U.79:     .LONG    <<BAD_VALUE-U.79>-4>         ;
     81ED  00098  ;TPA$TYPE
                    U.80:     .WORD    -32275                       ;
00000000V  0009A  ;TPA$ACTION
                    U.81:     .LONG    <<BAD_VALUE-U.81>-4>         ;
     85F7  0009E  ;TPA$TYPE
                    U.82:     .WORD    -31241                       ;
00000000V  000A0  ;TPA$ACTION
                    U.83:     .LONG    <<BAD_VALUE-U.83>-4>         ;
           000A4  ;SYNTAX
                    U.7:      .BLKB    0
     81F1  000A4  ;TPA$TYPE
                    U.84:     .WORD    -32271                       ;
```

```
                              00000000V 000A6 ;TPA$ACTION
                                              U.85:    .LONG   <<SYNTAX_ERR-U.85>-4>           ;
                                   81F0 000AA ;TPA$TYPE
                                              U.86:    .WORD   -32272                          ;
                              00000000V 000AC ;TPA$ACTION
                                              U.87:    .LONG   <<SYNTAX_ERR-U.87>-4>           ;
                                   81F6 000B0 ;TPA$TYPE
                                              U.88:    .WORD   -32266                          ;
                              00000000V 000B2 ;TPA$ACTION
                                              U.89:    .LONG   <<SYNTAX_ERR-U.89>-4>           ;
                                   81ED 000B6 ;TPA$TYPE
                                              U.90:    .WORD   -32275                          ;
                              00000000V 000B8 ;TPA$ACTION
                                              U.91:    .LONG   <<SYNTAX_ERR-U.91>-4>           ;
                                   85F7 000BC ;TPA$TYPE
                                              U.92:    .WORD   -31241                          ;
                              00000000V 000BE ;TPA$ACTION
                                              U.93:    .LONG   <<SYNTAX_ERR-U.93>-4>           ;

                                                       .PSECT  _LIB$KEY0$,NOWRT,  SHR,  PIC,1

                                        00000 KEY_TBL::
                                                       .BLKB   0
                                        00000 ;TPA$KEY0
                                              U.1:     .BLKB   0
                                        00000 NUMKEY_TBL::
                                                       .BLKB   0
                                        00000 ;TPA$KEY0
                                              U.43:    .BLKB   0

                                                       .PSECT  $PLIT$,NOWRT,NOEXE,2

                         52 45 54 46 41 00000 P.AAB:   .ASCII  \AFTER\                         ;
                                        00005          .BLKB   3
                              00000005  00008 P.AAA:   .LONG   5                               ;
                              00000000' 0000C          .ADDRESS P.AAB                          ;
      53 4B 43 4F 4C 42 5F 44 41 42     00010 P.AAD:   .ASCII  \BAD_BLOCKS\                    ;
                                        0001A          .BLKB   2
                              0000000A  0001C P.AAC:   .LONG   10                              ;
                              00000000' 00020          .ADDRESS P.AAD                          ;
                      45 52 4F 46 45 42 00024 P.AAF:   .ASCII  \BEFORE\                        ;
                                        0002A          .BLKB   2
                              00000006  0002C P.AAE:   .LONG   6                               ;
                              00000000' 00030          .ADDRESS P.AAF                          ;
                         45 4E 49 4C 24 00034 P.AAH:   .ASCII  \$LINE\                         ;
                                        00039          .BLKB   3
                              00000005  0003C P.AAG:   .LONG   5                               ;
                              00000000' 00040          .ADDRESS P.AAH                          ;
                         54 4E 55 4F 43 00044 P.AAJ:   .ASCII  \COUNT\                         ;
                                        00049          .BLKB   3
                              00000005  0004C P.AAI:   .LONG   5                               ;
                              00000000' 00050          .ADDRESS P.AAJ                          ;
                52 45 44 4E 49 4C 59 43 00054 P.AAL:   .ASCII  \CYLINDER\                      ;
                              00000008  0005C P.AAK:   .LONG   8                               ;
                              00000000' 00060          .ADDRESS P.AAL                          ;
                      45 43 49 56 45 44 00064 P.AAN:   .ASCII  \DEVICE\                        ;
                                        0006A          .BLKB   2
```

```
                                 00000006.  0006C P.AAM:   .LONG    6                  ;
                                 00000000'  00070          .ADDRESS P.AAN              ;
              45 53 49 43 52 45 58 45       00074 P.AAP:   .ASCII   \EXERCISE\         ;
                                 00000008.  0007C P.AAO:   .LONG    8                  ;
                                 00000000'  00080          .ADDRESS P.AAP              ;
                          4C 4C 55 46       00084 P.AAR:   .ASCII   \FULL\             ;
                                 00000004.  00088 P.AAQ:   .LONG    4                  ;
                                 00000000'  0008C          .ADDRESS P.AAR              ;
                          50 45 45 4B       00090 P.AAT:   .ASCII   \KEEP\             ;
                                 00000004.  00094 P.AAS:   .LONG    4                  ;
                                 00000000'  00098          .ADDRESS P.AAT              ;
                             47 4F 4C       0009C P.AAV:   .ASCII   \LOG\              ;
                                            0009F          .BLKB    1
                                 00000003.  000A0 P.AAU:   .LONG    3                  ;
                                 00000000'  000A4          .ADDRESS P.AAV              ;
  4B 43 4F 4C 42 5F 4C 41 43 49 47 4F 4C    000A8 P.AAX:   .ASCII   \LOGICAL_BLOCK\    ;
                                            000B5          .BLKB    3
                                 0000000D.  000B8 P.AAW:   .LONG    13                 ;
                                 00000000'  000BC          .ADDRESS P.AAX              ;
                       46 53 42 44 4D       000C0 P.AAZ:   .ASCII   \MDBSF\            ;
                                            000C5          .BLKB    3
                                 00000005.  000C8 P.AAY:   .LONG    5                  ;
                                 00000000'  000CC          .ADDRESS P.AAZ              ;
                    54 55 50 54 55 4F       000D0 P.ABB:   .ASCII   \OUTPUT\           ;
                                            000D6          .BLKB    2
                                 00000006.  000D8 P.ABA:   .LONG    6                  ;
                                 00000000'  000DC          .ADDRESS P.ABB              ;
                 4E 52 45 54 54 41 50       000E0 P.ABD:   .ASCII   \PATTERN\          ;
                                            000E7          .BLKB    1
                                 00000007.  000E8 P.ABC:   .LONG    7                  ;
                                 00000000'  000EC          .ADDRESS P.ABD              ;
  20 3D 20 53 4B 43 4F 4C 42 5F 44 41 42    000F0 P.ABF:   .ASCII   \BAD_BLOCKS = \    ;
                                            000FD          .BLKB    3
                                 0000000D.  00100 P.ABE:   .LONG    13                 ;
                                 00000000'  00104          .ADDRESS P.ABF              ;
                       59 52 54 45 52       00108 P.ABH:   .ASCII   \RETRY\            ;
                                            0010D          .BLKB    3
                                 00000005.  00110 P.ABG:   .LONG    5                  ;
                                 00000000'  00114          .ADDRESS P.ABH              ;
                       46 53 42 44 53       00118 P.ABJ:   .ASCII   \SDBSF\            ;
                                            0011D          .BLKB    3
                                 00000005.  00120 P.ABI:   .LONG    5                  ;
                                 00000000'  00124          .ADDRESS P.ABJ              ;
                    52 4F 54 43 45 53       00128 P.ABL:   .ASCII   \SECTOR\           ;
                                            0012E          .BLKB    2
                                 00000006.  00130 P.ABK:   .LONG    6                  ;
                                 00000000'  00134          .ADDRESS P.ABL              ;
                          57 4F 48 53       00138 P.ABN:   .ASCII   \SHOW\             ;
                                 00000004.  0013C P.ABM:   .LONG    4                  ;
                                 00000000'  00140          .ADDRESS P.ABN              ;
                       4B 43 41 52 54       00144 P.ABP:   .ASCII   \TRACK\            ;
                                            00149          .BLKB    3
                                 00000005.  0014C P.ABO:   .LONG    5                  ;
                                 00000000'  00150          .ADDRESS P.ABP              ;

                                                  AFTER=                  P.AAA
                                                  BAD_BLOCKS=             P.AAC
```

```
BEFORE=          P.AAE
COMMAND_LIN=     P.AAG
COUNT=           P.AAI
CYLINDER=        P.AAK
DEVICE=          P.AAM
EXERCISE=        P.AAO
FULL=            P.AAQ
KEEP=            P.AAS
LOG=             P.AAU
LOGICAL_BLK=     P.AAW
MDBSF=           P.AAY
OUTPUT=          P.ABA
PATTERN=         P.ABC
PROMPT=          P.ABE
RETRY=           P.ABG
SDBSF=           P.ABI
SECTOR=          P.ABK
SHOW_QUAL=       P.ABM
TRACK=           P.ABO
        .EXTRN  BAD$CHECK_LTK, BAD$CHECK_NLT
        .EXTRN  BAD$CVT_LTK_LONG
        .EXTRN  BAD$CVT_NLT_LONG
        .EXTRN  BAD$CVT_PHY_LOG
        .EXTRN  BAD$PREPARE_REPORT
        .EXTRN  BAD$PRODUCE_REPORT
        .EXTRN  BAD$SYNC_IO, CHECKSUM2
        .EXTRN  CLI$GET_VALUE, CLI$PRESENT
        .EXTRN  LIB$GET_INPUT, LIB$TPARSE
        .EXTRN  BAD$GL_BAD_TERM
        .EXTRN  BAD$GB_BLOCK_FACT
        .EXTRN  BAD$GL_BYTES_CYL
        .EXTRN  BAD$GL_BYTES_TRK
        .EXTRN  BAD$GL_CHAN, BAD$GA_COMND_LINE
        .EXTRN  BAD$GL_CONTEXT, BAD$GL_CYLINDERS
        .EXTRN  BAD$GA_BUFADR, BAD$GL_DEVCHAR
        .EXTRN  BAD$GL_DEVCLASS
        .EXTRN  BAD$GA_DEVICE, BAD$GA_DEVNAM
        .EXTRN  BAD$GL_DEVNAM, BAD$GQ_DEVNAM
        .EXTRN  BAD$GA_FILESPEC
        .EXTRN  BAD$GL_FUNC, BAD$GA_GETDVI
        .EXTRN  BAD$GA_INPUT_DESC
        .EXTRN  BAD$GQ_IOSB, BAD$GL_MAXBLOCK
        .EXTRN  BAD$GA_MDBSF, BAD$GL_MDBSF_PTR
        .EXTRN  BAD$GL_PAGCNT, BAD$GA_SDBSF
        .EXTRN  BAD$GL_SDBSF_PTR
        .EXTRN  BAD$GL_SECTORS, BAD$GL_SECTOR_SIZ
        .EXTRN  BAD$GL_SERIALNUM
        .EXTRN  BAD$GL_STATUS, BAD$GB_TERM_COUNT
        .EXTRN  BAD$GA_TPB, BAD$GL_TRACKS
        .EXTRN  BAD$GL_TRNSFR_CNT
        .EXTRN  BAD$_ALIGNDISK, BAD$_ASSIGN
        .EXTRN  BAD$_BADVALUE, BAD$_DEVNOTBLK
        .EXTRN  BAD$_DEVNOTFOR, BAD$_DEVRCT
        .EXTRN  BAD$_GETDVI, BAD$_INSVIRMEM
        .EXTRN  BAD$_INVALQUAL, BAD$_IVBLKNUM
        .EXTRN  BAD$_IVBLKENT, BAD$_LSTSTRING
        .EXTRN  BAD$_MDBSFCRUPT
```

```
                                                        .EXTRN  BAD$_MDBSFRFAIL
                                                        .EXTRN  BAD$_NOBADINFO, BAD$_NOOUTQUAL
                                                        .EXTRN  BAD$_REBLDWARN, BAD$_SDBSFFULL
                                                        .EXTRN  BAD$_SDBSFRFAIL
                                                        .EXTRN  SYS$GETDVIW, SYS$ASSIGN
                                                        .WEAK   BAD$STA_INIT, BAD$STR_TRIM
                                                        .WEAK   BAD$VALIDATE_PACK

                                                        .PSECT  $CODE$,NOWRT,2

                                       0FFC 00000       .ENTRY  BAD$INIT, Save R2,R3,R4,R5,R6,R7,R8,R9,R10,-;  0456
                                                                R11                                         :
                  5B 00000000G  00   9E 00002           MOVAB   CLI$GET_VALUE, R11                          :
                  5A     0000G  CF   9E 00009           MOVAB   BAD$GA_DEVICE, R10                          :
                  59 00000000G  00   9E 0000E           MOVAB   LIB$STOP, R9                                :
                  58     0000'  CF   9E 00015           MOVAB   BAD_BLOCKS, R8                              :
                  57 00000000G  00   9E 0001A           MOVAB   CLI$PRESENT, R7                             :
                  56     0000G  CF   9E 00021           MOVAB   BAD$GL_CONTEXT, R6                          :
                  66            10   88 00026           BISB2   #16, BAD$GL_CONTEXT                         ;  0482
   08       00    6E            00   2C 00029           MOVC5   #0, (SP), #0, #8, BAD$GA_COMND_LINE         ;  0487
                         0000G  CF      0002E                                                               :
   08       00    6E            00   2C 00031           MOVC5   #0, (SP), #0, #8, BAD$GA_DEVICE             ;  0488
                                6A      00036                                                               :
   08       00    6E            00   2C 00037           MOVC5   #0, (SP), #0, #8, BAD$GA_INPUT_DESC         ;  0489
                         0000G  CF      0003C                                                               :
   08       00    6E            00   2C 0003F           MOVC5   #0, (SP), #0, #8, BAD$GA_FILESPEC           ;  0490
                         0000G  CF      00044                                                               :
             0000G  CF          02   90 00047           MOVB    #2, BAD$GA_COMND_LINE+3                     ;  0491
                03  AA          02   90 0004C           MOVB    #2, BAD$GA_DEVICE+3                         ;  0492
             0000G  CF          02   90 00050           MOVB    #2, BAD$GA_INPUT_DESC+3                     ;  0493
             0000G  CF          02   90 00055           MOVB    #2, BAD$GA_FILESPEC+3                       ;  0494
                    50   0000G  CF   9E 0005A           MOVAB   BAD$STA_INIT, R0                            ;  0496
                                05   13 0005F           BEQL    1$                                          :
             0000G  CF          00   FB 00061           CALLS   #0, BAD$STA_INIT                            ;  0497
                         0000G  CF   9F 00066  1$:      PUSHAB  BAD$GA_COMND_LINE                           ;  0502
                            20  A8   9F 0006A           PUSHAB  COMMAND_LIN                                 :
                    6B          02   FB 0006D           CALLS   #2, CLI$GET_VALUE                           :
                    50   0000G  CF   9E 00070           MOVAB   BAD$STR_TRIM, R0                            ;  0503
                                52   D4 00075           CLRL    R2                                          :
                                50   D5 00077           TSTL    R0                                          :
                                0B   13 00079           BEQL    2$                                          :
                                52   D6 0007B           INCL    R2                                          :
                         0000G  CF   9F 0007D           PUSHAB  BAD$GA_COMND_LINE                           ;  0504
             0000G  CF          01   FB 00081           CALLS   #1, BAD$STR_TRIM                            :
                                5A   DD 00086  2$:      PUSHL   R10                                         ;  0506
                            50  A8   9F 00088           PUSHAB  DEVICE                                      :
                    6B          02   FB 0008B           CALLS   #2, CLI$GET_VALUE                           :
                    07          52   E9 0008E           BLBC    R2, 3$                                      ;  0507
                                5A   DD 00091           PUSHL   R10                                         ;  0508
             0000G  CF          01   FB 00093           CALLS   #1, BAD$STR_TRIM                            :
                                7E   7C 00098  3$:      CLRQ    -(SP)                                       ;  0512
                                7E   7C 0009A           CLRQ    -(SP)                                       :
                         0000G  CF   9F 0009C           PUSHAB  BAD$GA_GETDVI                               :
                                5A   DD 000A0           PUSHL   R10                                         :
                                7E   7C 000A2           CLRQ    -(SP)                                       :
         00000000G  00          08   FB 000A4           CALLS   #8, SYS$GETDVIW                             :
             0000G  CF          50   D0 000AB           MOVL    R0, BAD$GL_STATUS                           :
```

```
                           11                     50  E8  000B0         BLBS    R0, 4$                                      ;
                                        0000G     CF  DD  000B3         PUSHL   BAD$GL_STATUS                               ; 0514
                                                  5A  DD  000B7         PUSHL   R10                                         ;
                                                  01  DD  000B9         PUSHL   #1                                          ;
                                    00000000G     8F  DD  000BB         PUSHL   #BAD$_GETDVI                                ;
                           69                     04  FB  000C1         CALLS   #4, LIB$STOP                                ;
                           01           0000G     CF  D1  000C4   4$:   CMPL    BAD$GL_DEVCLASS, #1                         ; 0516
                                                  0D  13  000C9         BEQL    5$                                          ;
                                                  5A  DD  000CB         PUSHL   R10                                         ; 0518
                                                  01  DD  000CD         PUSHL   #1                                          ;
                                    00000000G     8F  DD  000CF         PUSHL   #BAD$_DEVNOTBLK                             ;
                           69                     03  FB  000D5         CALLS   #3, LIB$STOP                                ;
                           0D           0000G     CF  E9  000D8   5$:   BLBC    BAD$GL_DEVCHAR+1, 6$                        ; 0520
                                                  5A  DD  000DD         PUSHL   R10                                         ; 0522
                                                  01  DD  000DF         PUSHL   #1                                          ;
                                    00000000G     8F  DD  000E1         PUSHL   #BAD$_DEVRCT                                ;
                           69                     03  FB  000E7         CALLS   #3, LIB$STOP                                ;
                           14           0000G     CF  E8  000EA   6$:   BLBS    BAD$GL_DEVCHAR+3, 7$                        ; 0524
                           50           0000G     CF  9E  000EF         MOVAB   BAD$STA_INIT, R0                            ; 0525
                                                  0D  12  000F4         BNEQ    7$                                          ;
                                                  5A  DD  000F6         PUSHL   R10                                         ; 0527
                                                  01  DD  000F8         PUSHL   #1                                          ;
                                    00000000G     8F  DD  000FA         PUSHL   #BAD$_DEVNOTFOR                             ;
                           69                     03  FB  00100         CALLS   #3, LIB$STOP                                ;
                                                  7E  7C  00103   7$:   CLRQ    -(SP)                                       ; 0531
                                        0000G     CF  9F  00105         PUSHAB  BAD$GL_CHAN                                 ;
                                                  5A  DD  00109         PUSHL   R10                                         ;
                 00000000G     00                 04  FB  0010B         CALLS   #4, SYS$ASSIGN                              ;
                     0000G     CF                 50  D0  00112         MOVL    R0, BAD$GL_STATUS                           ;
                               11                 50  E8  00117         BLBS    R0, 8$                                      ;
                                        0000G     CF  DD  0011A         PUSHL   BAD$GL_STATUS                               ; 0533
                                                  5A  DD  0011E         PUSHL   R10                                         ;
                                                  01  DD  00120         PUSHL   #1                                          ;
                                    00000000G     8F  DD  00122         PUSHL   #BAD$_ASSIGN                                ;
                           69                     04  FB  00128         CALLS   #4, LIB$STOP                                ;
                           50           0000G     CF  9E  0012B   8$:   MOVAB   BAD$VALIDATE_PACK, R0                       ; 0535
                                                  05  13  00130         BEQL    9$                                          ;
                     0000G     CF                 00  FB  00132         CALLS   #0, BAD$VALIDATE_PACK                       ; 0536
                     0000G     CF       0000G     CF  B0  00137   9$:   MOVW    BAD$GL_DEVNAM, BAD$GQ_DEVNAM                ; 0538
                     0000G     CF        020E     8F  B0  0013E         MOVW    #526, BAD$GQ_DEVNAM+2                       ; 0539
                     0000G     CF       0000G     CF  9E  00145         MOVAB   BAD$GA_DEVNAM, BAD$GQ_DEVNAM+4              ; 0541
                        52              0000G     CF  D0  0014C         MOVL    BAD$GL_MAXBLOCK, R2                         ; 0543
                 00001000      8F                 52  D1  00151         CMPL    R2, #4096                                   ;
                                                  67  15  00158         BLEQ    11$                                         ;
                        01     A6                 01  88  0015A         BISB2   #1, BAD$GL_CONTEXT+1                        ; 0546
            50       0000G     CF       0000G     CF  C5  0015E         MULL3   BAD$GL_CYLINDERS, BAD$GL_TRACKS, R0         ; 0548
                               51       0000G     CF  D0  00166         MOVL    BAD$GL_SECTORS, R1                          ; 0549
                               50                 51  C4  0016B         MULL2   R1, R0                                      ;
            53                 50                 52  C7  0016E         DIVL3   R2, R0, R3                                  ; 0550
                     0000G     CF                 53  90  00172         MOVB    R3, BAD$GB_BLOCK_FACT                       ;
                               50       0000G     CF  9A  00177         MOVZBL  BAD$GB_BLOCK_FACT, R0                       ; 0553
            50                 51                 50  C7  0017C         DIVL3   R0, R1, R0                                  ;
   0000G    CF                 52                 50  C3  00180         SUBL3   R0, R2, BAD$GL_MDBSF_PTR                    ; 0552
                               50       0000G     CF  9A  00186         MOVZBL  BAD$GB_BLOCK_FACT, R0                       ; 0554
   0000G    CF  00000200       8F                 50  C7  0018B         DIVL3   R0, #512, BAD$GL_SECTOR_SIZ                 ;
   0000G    CF                 51       0000G     CF  C5  00195         MULL3   BAD$GL_SECTOR_SIZ, R1, BAD$GL_BYTES_TRK     ; 0555
   0000G    CF       0000G     CF       0000G     CF  C5  0019D         MULL3   BAD$GL_TRACKS, BAD$GL_BYTES_TRK, -          ; 0556
```

```
                                                                BAD$GL_BYTES_CYL                          ;
                       50      0000G  CF  D0 001A7         MOVL    BAD$GL_BYTES_CYL, R0                   ; 0557
            0000FE00   8F             50  D1 001AC         CMPL    R0, #65024                             ;
                                      05  1B 001B3         BLEQU   10$                                    ;
                       50       FE00  8F  3C 001B5         MOVZWL  #65024, R0                             ;
                0000G  CF             50  D0 001BA  10$:   MOVL    R0, BAD$GL_TRNSFR_CNT                  ;
                                      07  11 001BF         BRB     12$                                    ; 0543
                0000G  CF       FE00  8F  3C 001C1  11$:   MOVZWL  #65024, BAD$GL_TRNSFR_CNT              ; 0561
        0000G  CF      0000G  CF 00000200  8F  C7 001C8  12$:   DIVL3   #512, BAD$GL_TRNSFR_CNT, BAD$GL_PAGCNT ; 0563
                                      58  DD 001D4         PUSHL   R8                                     ; 0577
                       67             01  FB 001D6         CALLS   #1, CLI$PRESENT                        ;
     66         01     00             50  F0 001D9         INSV    R0, #0, #1, BAD$GL_CONTEXT             ;
                                60    A8  9F 001DE         PUSHAB  EXERCISE                               ; 0578
                       67             01  FB 001E1         CALLS   #1, CLI$PRESENT                        ;
     66         01     02             50  F0 001E4         INSV    R0, #2, #1, BAD$GL_CONTEXT             ;
                                78    A8  9F 001E9         PUSHAB  KEEP                                   ; 0579
                       67             01  FB 001EC         CALLS   #1, CLI$PRESENT                        ;
     66         01     06             50  F0 001EF         INSV    R0, #6, #1, BAD$GL_CONTEXT             ;
                              0084    C8  9F 001F4         PUSHAB  LOG                                    ; 0580
                       67             01  FB 001F8         CALLS   #1, CLI$PRESENT                        ;
     66         01     07             50  F0 001FB         INSV    R0, #7, #1, BAD$GL_CONTEXT             ;
                              00BC    C8  9F 00200         PUSHAB  OUTPUT                                 ; 0581
                       67             01  FB 00204         CALLS   #1, CLI$PRESENT                        ;
 01  A6         01     01             50  F0 00207         INSV    R0, #1, #1, BAD$GL_CONTEXT+1           ;
                              00F4    C8  9F 0020D         PUSHAB  RETRY                                  ; 0582
                       67             01  FB 00211         CALLS   #1, CLI$PRESENT                        ;
 01  A6         01     03             50  F0 00214         INSV    R0, #3, #1, BAD$GL_CONTEXT+1           ;
                              0120    C8  9F 0021A         PUSHAB  SHOW_QUAL                              ; 0583
                       67             01  FB 0021E         CALLS   #1, CLI$PRESENT                        ;
 01  A6         01     04             50  F0 00221         INSV    R0, #4, #1, BAD$GL_CONTEXT+1           ;
                0000G  CF             21  D0 00227         MOVL    #33, BAD$GL_FUNC                       ; 0596
                       21         01  A6  E9 0022C         BLBC    BAD$GL_CONTEXT+1, 13$                  ; 0597
                              0000G   CF  9F 00230         PUSHAB  BAD$GA_MDBSF                           ; 0600
                       7E      0400   8F  3C 00234         MOVZWL  #1024, -(SP)                           ;
                0000V  CF             02  FB 00239         CALLS   #2, BAD$ALLOC_MEM                      ;
        0000G  CF      0000G  CF 00000200  8F  C1 0023E         ADDL3   #512, BAD$GA_MDBSF, BAD$GA_SDBSF      ; 0601
                0000V  CF             00  FB 0024A         CALLS   #0, GET_MDBSF                          ; 0602
                                      0E  11 0024F         BRB     14$                                    ; 0597
                              0000G   CF  9F 00251  13$:   PUSHAB  BAD$GA_SDBSF                           ; 0605
                       7E      0200   8F  3C 00255         MOVZWL  #512, -(SP)                            ;
                0000V  CF             02  FB 0025A         CALLS   #2, BAD$ALLOC_MEM                      ;
                16        01   A6     01  E1 0025F  14$:   BBC     #1, BAD$GL_CONTEXT+1, 15$              ; 0626
                              0000G   CF  9F 00264         PUSHAB  BAD$GA_FILESPEC                        ; 0628
                              00BC    C8  9F 00268         PUSHAB  OUTPUT                                 ;
                       6B             02  FB 0026C         CALLS   #2, CLI$GET_VALUE                      ;
            00000000G  00             00  FB 0026F         CALLS   #0, BAD$PREPARE_REPORT                 ; 0629
                   01  A6             20  88 00276         BISB2   #32, BAD$GL_CONTEXT+1                  ; 0630
                63        01   A6     04  E1 0027A  15$:   BBC     #4, BAD$GL_CONTEXT+1, 22$              ; 0652
                17        01   A6     01  E0 0027F         BBS     #1, BAD$GL_CONTEXT+1, 16$              ; 0654
                              0000G   CF  9F 00284         PUSHAB  BAD$GA_COMND_LINE                      ; 0656
                                      01  DD 00288         PUSHL   #1                                     ;
                          00000000G   8F  DD 0028A         PUSHL   #BAD$_LSTSTRING                        ;
                                      7E  D4 00290         CLRL    -(SP)                                  ;
                          00000000G   8F  DD 00292         PUSHL   #BAD$_NOOUTQUAL                        ;
                       69             05  FB 00298         CALLS   #5, LIB$STOP                           ;
                                10    A8  9F 0029B  16$:   PUSHAB  BEFORE                                 ; 0658
                       67             01  FB 0029E         CALLS   #1, CLI$PRESENT                        ;
```

```
 01   A6      01           06            50  F0 002A1          INSV    R0, #6, #1, BAD$GL_CONTEXT+1               ;
                           34            50  E9 002A7          BLBC    R0, 21$                                    ;
                           07     01     A6  E9 002AA          BLBC    BAD$GL_CONTEXT+1, 17$                      ; 0661
                   0000V   CF            00  FB 002AE          CALLS   #0, GET_SDBSF                              ; 0662
                           0B            11  002B3             BRB     18$                                        ;
                   0000V   CF            00  FB 002B5 17$:     CALLS   #0, GET_NLT_SDBSF                          ; 0663
 01   A6      01           07            50  F0 002BA          INSV    R0, #7, #1, BAD$GL_CONTEXT+1               ;
                           05     01     A6  E8 002C0 18$:     BLBS    BAD$GL_CONTEXT+1, 19$                      ; 0664
                                  01     A6  95 002C4          TSTB    BAD$GL_CONTEXT+1                           ;
                                         07  18 002C7          BGEQ    20$                                        ;
               00000000G   00            00  FB 002C9 19$:     CALLS   #0, BAD$PRODUCE_REPORT                     ; 0665
                                  EC     A8  9F 002D0 20$:     PUSHAB  AFTER                                      ; 0666
                           67            01  FB 002D3          CALLS   #1, CLI$PRESENT                            ;
 01   A6      01           05            50  F0 002D6          INSV    R0, #5, #1, BAD$GL_CONTEXT+1               ;
                                         04  11 002DC          BRB     22$                                        ; 0658
                      01   A6            20  88 002DE 21$:     BISB2   #32, BAD$GL_CONTEXT+1                      ; 0669
              09           66            06  E0 002E2 22$:     BBS     #6, BAD$GL_CONTEXT, 23$                    ; 0693
              17           66            02  E0 002E6          BBS     #2, BAD$GL_CONTEXT, 25$                    ; 0694
              12      01   A6            06  E0 002EA          BBS     #6, BAD$GL_CONTEXT+1, 25$                  ;
                           07     01     A6  E9 002EF 23$:     BLBC    BAD$GL_CONTEXT+1, 24$                      ; 0696
                   0000V   CF            00  FB 002F3          CALLS   #0, GET_SDBSF                              ; 0697
                                         0C  11 002F8          BRB     26$                                        ;
                   0000V   CF            00  FB 002FA 24$:     CALLS   #0, GET_NLT_SDBSF                          ; 0698
                                         05  11 002FF          BRB     26$                                        ; 0696
                   0000V   CF            00  FB 00301 25$:     CALLS   #0, INIT_SDBSF                             ; 0700
                           4F            66  E9 00306 26$:     BLBC    BAD$GL_CONTEXT, 31$                        ; 0736
                                         52  D4 00309          CLRL    VALUE_PRESENT                              ; 0741
                               0000G     CF  9F 0030B 27$:     PUSHAB  BAD$GA_INPUT_DESC                          ; 0742
                                         58  DD 0030F          PUSHL   R8                                         ;
                           6B            02  FB 00311          CALLS   #2, CLI$GET_VALUE                          ;
                   0000G   CF            50  D0 00314          MOVL    R0, BAD$GL_STATUS                          ;
                           0A            50  E9 00319          BLBC    R0, 28$                                    ;
                           52            01  D0 0031C          MOVL    #1, VALUE_PRESENT                          ; 0744
                   0000V   CF            00  FB 0031F          CALLS   #0, PROCESS_BADBLOCKS                      ; 0745
                                         E5  11 00324          BRB     27$                                        ; 0742
                           2C            52  E8 00326 28$:     BLBS    VALUE_PRESENT, 30$                         ; 0748
                           66            20  88 00329          BISB2   #32, BAD$GL_CONTEXT                        ; 0751
                               00E4      C8  9F 0032C 29$:     PUSHAB  PROMPT                                     ; 0752
                               0000G     CF  9F 00330          PUSHAB  BAD$GA_INPUT_DESC                          ;
               00000000G   00            02  FB 00334          CALLS   #2, LIB$GET_INPUT                          ;
                   0000G   CF            50  D0 0033B          MOVL    R0, BAD$GL_STATUS                          ;
                           12            50  E9 00340          BLBC    R0, 30$                                    ;
               0001827A    8F  0000G     CF  D1 00343          CMPL    BAD$GL_STATUS, #98938                      ; 0755
                                         07  13 0034C          BEQL    30$                                        ;
                   0000V   CF            00  FB 0034E          CALLS   #0, PROCESS_BADBLOCKS                      ; 0757
                                         D7  11 00353          BRB     29$                                        ; 0752
                           66            20  8A 00355 30$:     BICB2   #32, BAD$GL_CONTEXT                        ; 0760
              50           66            02  E1 00358 31$:     BBC     #2, BAD$GL_CONTEXT, 33$                    ; 0790
                                  6C     A8  9F 0035C          PUSHAB  FULL                                       ; 0792
                           67            01  FB 0035F          CALLS   #1, CLI$PRESENT                            ;
      66      01           03            50  F0 00362          INSV    R0, #3, #1, BAD$GL_CONTEXT                 ;
                      01   A6     80     8F  88 00367          BISB2   #128, BAD$GL_CONTEXT+1                     ; 0793
                               00CC      C8  9F 0036C          PUSHAB  PATTERN                                    ; 0795
                           67            01  FB 00370          CALLS   #1, CLI$PRESENT                            ;
 01   A6      01           02            50  F0 00373          INSV    R0, #2, #1, BAD$GL_CONTEXT+1               ;
                           05            50  E9 00379          BLBC    R0, 32$                                    ;
                   0000V   CF            00  FB 0037C          CALLS   #0, GET_PATTERN                            ; 0797
```

```
                               0000G   CF   9F 00381 32$:    PUSHAB  BAD$GA_BUFADR                              ; 0800
        7E      0000G   CF              0A   78 00385          ASHL    #10, BAD$GL_PAGCNT, -(SP)                  ; 0799
                0000V   CF              02   FB 0038B          CALLS   #2, BAD$ALLOC_MEM                          ;
        50      0000G   CF              09   78 00390          ASHL    #9, BAD$GL_PAGCNT, R0                      ; 0803
                0000G   CF     0000GDF40     9E 00396          MOVAB   @BAD$GA_BUFADR[R0], BAD$GA_BUFADR+4        ;
                               0000G   CF   9F 0039E          PUSHAB  BAD$GA_TPB                                 ; 0806
                        7E     0200    8F   3C 003A2          MOVZWL  #512, -(SP)                                ;
                0000V   CF              02   FB 003A7          CALLS   #2, BAD$ALLOC_MEM                          ;
                        66              10   8A 003AC 33$:    BICB2   #16, BAD$GL_CONTEXT                        ; 0812
                                             04 003AF          RET                                                ; 0814

; Routine Size:  944 bytes,    Routine Base:  $CODE$ + 0000

;  705         0815  1
```

```
;  707      0816  1 %SBTTL  'bad$alloc_mem -- Allocate Memory'
;  708      0817  1 GLOBAL ROUTINE  bad$alloc_mem (byte_count, return_address) : NOVALUE =
;  709      0818  1 !++
;  710      0819  1 !
;  711      0820  1 ! Functional Description:
;  712      0821  1 !
;  713      0822  1 !     This routine will allocate page(s) of memory.  If an error occurs while
;  714      0823  1 !     attempting to allocate the memory, the error will be signalled FATAL.
;  715      0824  1 !
;  716      0825  1 ! Inputs:
;  717      0826  1 !
;  718      0827  1 !     byte_count          VALUE   Number of bytes to allocate.
;  719      0828  1 !
;  720      0829  1 ! Outputs:
;  721      0830  1 !
;  722      0831  1 !     return_address      ADDRESS Address of the location to store the
;  723      0832  1 !                                 beginning address of allocated memory.
;  724      0833  1 !
;  725      0834  1 ! Side Effects:
;  726      0835  1 !
;  727      0836  1 !     If there is an error in allocating memory, this procedure will terminate
;  728      0837  1 !     the analysis via a SIGNAL_STOP.
;  729      0838  1 !
;  730      0839  1 !--
;  731      0840  2 BEGIN
;  732      0841  2 MAP
;  733      0842  2     return_address : REF $BBLOCK;                     ! Point to return address.
;  734      0843  2
;  735      0844  2 LOCAL
;  736      0845  2     addr_vec : VECTOR [2, LONG],
;  737      0846  2     page_count;
;  738      0847  2
;  739      0848  2 IF (page_count = .byte_count / bad$k_page_size) LSS 1
;  740      0849  2 THEN    page_count = 1;
;  741      0850  2
;  742    P 0851  3 IF NOT (bad$gl_status = $EXPREG (PAGCNT = .page_count,
;  743      0852  3                           RETADR = addr_vec))      ! Allocate the requested amount
;  744      0853  2 THEN                                               ! of storage.
;  745      0854  2     SIGNAL_STOP (bad$_insvirmem, 0, .bad$gl_status);
;  746      0855  2
;  747      0856  2 .return_address = .addr_vec [0];
;  748      0857  2
;  749      0858  1 END;    ! Of GLOBAL ROUTINE bad$alloc_mem


                                                       .EXTRN  SYS$EXPREG

                                      0000 00000       .ENTRY  BAD$ALLOC_MEM, Save nothing          ; 0817
                    5E                 08  C2 00002       SUBL2   #8, SP                               ;
    50        04    AC 00000200   8F  C7 00005       DIVL3   #512, BYTE_COUNT, PAGE_COUNT          ; 0848
                                       03  14 0000E       BGTR    1$                                   ;
                    50                 01  D0 00010       MOVL    #1, PAGE_COUNT                       ; 0849
                                       7E  7C 00013 1$:   CLRQ    -(SP)                                ; 0852
                        08             AE  9F 00015       PUSHAB  ADDR_VEC                             ;
                                       50  DD 00018       PUSHL   PAGE_COUNT                           ;
       00000000G  00                   04  FB 0001A       CALLS   #4, SYS$EXPREG                       ;
```

```
                 0000G  CF          50  D0 00021         MOVL    R0, BAD$GL_STATUS              :
                        13          50  E8 00026         BLBS    R0, 2$                         :
                               0000G CF  DD 00029         PUSHL   BAD$GL_STATUS                  : 0854
                                    7E  D4 0002D         CLRL    -(SP)                          :
                           00000000G 8F  DD 0002F         PUSHL   #BAD$_INSVIRMEM                :
            00000000G  00               03  FB 00035         CALLS   #3, LIB$STOP                   :
                   08  BC               6E  D0 0003C 2$:     MOVL    ADDR_VEC, @RETURN_ADDRESS      : 0856
                                        04 00040         RET                                    : 0858

; Routine Size:  65 bytes,    Routine Base:  $CODE$ + 03B0

;  750          0859  1
```

```
%SBTTL  'get_mdbsf -- Get a valid copy of the MDBSF'
ROUTINE get_mdbsf : NOVALUE =
!++
!
! Functional Description:
!
!     Obtain a valid copy of the Manufacturer's Detected Bad Sector File.
!
! Implicit Inputs:
!
!     bad$ga_mdbsf        the address of the MDBSF buffer.
!     bad$ga_sdbsf        the address of the SDBSF buffer.
!     bad$gl_mdbsf_ptr    the block number of the first block of the last track.
!
! Side Effects:
!
!     If unable to read the MDBSF, we will recreate BOTH the MDBSF and SDBSF
!     ONLY if we are going to exercise the medium and the user specified NOKEEP.
!     If he specified /EXERCISE=KEEP, we will abort the analysis.
!
!     If the request was something like /NOEXERCISE /OUTPUT we will attempt to
!     read a valid non last track format SDBSF.  Failing this we will
!     SIGNAL_STOP to inform the user of the situation.  If we were successful
!     in reading the non last track format SDBSF, we will convert the analysis
!     context to non last track by clearing the [ctx_v_ltdevice] context bit.
!     Thus allowing a listing to be produced.
!
!     In order to be able to exercise the media or update the bad block file,
!     the user must execute ANALYZE/MEDIA explicitly specifing /EXERCISE=NOKEEP.
!     This will convert the device to last track format and implies that the
!     user knows that data will be lost and the bad block files will be rebuilt.
!
!--
BEGIN
LOCAL
    first_found,
    value_present;

first_found = FALSE;                                        ! Indicator for buffer usage (dup copies of MDBSF).
value_present = FALSE;                                      ! Indicate whether we found a valid MDBSF.
INCR block FROM 0 TO 9 DO                                   ! Search the first 10 blocks of the
    BEGIN                                                   ! last track for the MDBSF.  Once found
    IF bad$gl_status = bad$sync_io (IF NOT .first_found     ! a second copy is obtained to validate
                            THEN    .bad$ga_mdbsf           ! the contents (since no checksum is
                            ELSE    .bad$ga_sdbsf,          ! available).
                            bad$k_page_size,
                            .bad$gl_mdbsf_ptr + .block)
    THEN
        BEGIN
        IF NOT .first_found                                 ! If this is the first time an MDBSF
        THEN                                                ! was found, check that it is not full,
            BEGIN                                           ! switch to second buffer for verfication.
            IF .(.bad$ga_mdbsf + (bad$k_page_size - 4)) EQL -1
            THEN    first_found = TRUE;                     ! Indicate that we found one.
            END
        ELSE
            BEGIN                                           ! Validate the MDBSF copies.
```

```
5               IF CH$EQL (bad$k_page_size, .bad$ga_mdbsf,  ! they should be identical.
5               bad$k_page_size, .bad$ga_sdbsf,0)
5               THEN
6                   BEGIN
6                   value_present = TRUE;                   ! We have a valid copy of the MDBSF,
6                   EXITLOOP;                               ! go finish the validation of the
5                   END;                                    ! pack.
4               END;
3           END;
2       END;
2
2 IF NOT .value_present                                     ! If we did not find a valid MDBSF,
2 THEN                                                      ! see if we can either rebuild a new
2     IF NOT recover_dbsfs ()                               ! set of bad block flies or recover
2     THEN                                                  ! a non lasttrack SDBSF.  If we can't,
2         SIGNAL_STOP (bad$_mdbsfrfail, 1, bad$ga_device);! then abort the analysis.
2
2 IF NOT .bad$gl_context [ctx_v_ltdevice]                   ! Did we recover a non last track SDBSF?
2 THEN
2     RETURN;                                               ! Yes, bypass the pack validation.
2
2 !
2 ! Validate the pack...
2 !
2 IF .bad$ga_mdbsf [ltk_w_cartridge] EQLU %X'FFFF'          ! Assure that it's not an
2 THEN                                                      ! alignment cartridge.
2     SIGNAL_STOP (bad$_aligndisk, 1, bad$ga_device)
2 ELSE
3     BEGIN                                                 ! Has the header been corrrupted?
3     IF (.bad$ga_mdbsf [ltk_l_serial_number] EQL 0) AND    ! It should have a serial number
4        (.bad$ga_mdbsf [ltk_w_unused] NEQ 0)               ! followed by a zero word, and the
3     THEN                                                  ! cartridge id (0's-data packs, 1's-diag pack)
3         IF .bad$gl_context [ctx_v_exercise]               ! in the header.
3         THEN
3             rebuild_dbsfs ()                              ! Rebuild the MDBSF and SDBSF if we can.
3         ELSE
3             SIGNAL_STOP (bad$_mdbsfcrupt, 1, bad$ga_device);
2     END;
2
1 END;      ! of ROUTINE get_mdbsf


                                  03FC 00000 GET_MDBSF:
                                             .WORD    Save R2,R3,R4,R5,R6,R7,R8,R9           ; 0861
        59     0000G  CF  9E 00002           MOVAB    BAD$GA_DEVICE, R9                       ;
        58 00000000G  00  9E 00007           MOVAB    LIB$STOP, R8                            ;
        57     0000G  CF  9E 0000E           MOVAB    BAD$GA_MDBSF, R7                        ;
                      55  7C 00013           CLRQ     VALUE_PRESENT                           ; 0899
                      54  D4 00015           CLRL     BLOCK                                   ; 0900
               0000GDF44  9F 00017 1$:       PUSHAB   @BAD$GL_MDBSF_PTR[BLOCK]                ; 0906
        7E     0200   8F  3C 0001C           MOVZWL   #512, -(SP)                             ; 0902
        04            56  E8 00021           BLBS     FIRST_FOUND, 2$                         ;
                      67  DD 00024           PUSHL    BAD$GA_MDBSF                            ; 0903
                      04  11 00026           BRB      3$                                      ;
```

```
                                0000G  CF   DD 00028 2$:    PUSHL   BAD$GA_SDBSF                                    ; 0904
                 00000000G  00          03   FB 0002C 3$:    CALLS   #3, BAD$SYNC_IO                                 ; 0902
                     0000G  CF          50   D0 00033        MOVL    R0, BAD$GL_STATUS                               ;
                            26          50   E9 00038        BLBC    R0, 5$                                          ;
                            13          56   E8 0003B        BLBS    FIRST_FOUND, 4$                                 ; 0912
                            50          67   D0 0003E        MOVL    BAD$GA_MDBSF, R0                                ;
                 FFFFFFFF   8F   01FC   C0   D1 00041        CMPL    508(R0), #-1                                    ;
                                        15   12 0004A        BNEQ    5$                                              ;
                            56          01   D0 0004C        MOVL    #1, FIRST_FOUND                                 ; 0913
                                        10   11 0004F        BRB     5$                                              ; 0909
       0000G  DF      00    B7   0200   8F   29 00051 4$:    CMPC3   #512, @BAD$GA_MDBSF, @BAD$GA_SDBSF              ; 0917
                                        05   12 0005A        BNEQ    5$                                              ;
                            55          01   D0 0005C        MOVL    #1, VALUE_PRESENT                               ; 0921
                                        04   11 0005F        BRB     6$                                              ; 0920
              B2            54          09   F3 00061 5$:    AOBLEQ  #9, BLOCK, 1$                                   ; 0900
                            15          55   E8 00065 6$:    BLBS    VALUE_PRESENT, 7$                               ; 0928
                     0000V  CF          00   FB 00068        CALLS   #0, RECOVER_DBSFS                               ; 0930
                            0D          50   E8 0006D        BLBS    R0, 7$                                          ;
                                        59   DD 00070        PUSHL   R9                                              ; 0932
                                        01   DD 00072        PUSHL   #1                                              ;
                           00000000G    8F   DD 00074        PUSHL   #BAD$_MDBSFRFAIL                                ;
                            68          03   FB 0007A        CALLS   #3, LIB$STOP                                    ;
                            39   0000G  CF   E9 0007D 7$:    BLBC    BAD$GL_CONTEXT+1, 11$                           ; 0934
                            52          67   D0 00082        MOVL    BAD$GA_MDBSF, R2                                ; 0941
                     FFFF   8F     06   A2   B1 00085        CMPW    6(R2), #65535                                   ;
                                        0C   12 0008B        BNEQ    8$                                              ;
                                        59   DD 0008D        PUSHL   R9                                              ; 0943
                                        01   DD 0008F        PUSHL   #1                                              ;
                           00000000G    8F   DD 00091        PUSHL   #BAD$_ALIGNDISK                                 ;
                                        1F   11 00097        BRB     10$                                             ;
                                        62   D5 00099 8$:    TSTL    (R2)                                            ; 0946
                                        1E   12 0009B        BNEQ    11$                                             ;
                                   04   A2   B5 0009D        TSTW    4(R2)                                           ; 0947
                                        19   13 000A0        BEQL    11$                                             ;
              06     0000G  CF          02   E1 000A2        BBC     #2, BAD$GL_CONTEXT, 9$                          ; 0949
                     0000V  CF          00   FB 000A8        CALLS   #0, REBUILD_DBSFS                               ; 0951
                                             04 000AD        RET                                                     ;
                                        59   DD 000AE 9$:    PUSHL   R9                                              ; 0953
                                        01   DD 000B0        PUSHL   #1                                              ;
                           00000000G    8F   DD 000B2        PUSHL   #BAD$_MDBSFCRUPT                                ;
                            68          03   FB 000B8 10$:   CALLS   #3, LIB$STOP                                    ;
                                             04 000BB 11$:   RET                                                     ; 0956

; Routine Size:  188 bytes,    Routine Base:  $CODE$ + 03F1

;  849          0957  1
```

```
1 %SBTTL  'recover_dbsfs -- Attempt to rebuild/recover an SDBSF'
1 ROUTINE recover_dbsfs =
1 !++
1 !
1 ! Functional Description:
1 !
1 !     A valid MDBSF was not found.  Check to see if the analysis context
1 !     will allow either of the following actions to be performed.
1 !
1 !      1. Attempt to rebuild both the MDBSF and the SDBSF if we are going to
1 !         exercise the medium and not save the old bad block data.
1 !
1 !      2. Attempt to find the "old" non last track format SDBSF if the medium
1 !         is not going to be exercised and only a listing type of action
1 !         was requested.  If successful, then we will change the context to
1 !         non last track.
1 !
1 ! Routine Value:
1 !
1 !     TRUE        If we were successful in rebuilding the MDBSF and SDBSF or
1 !                 in locating the "old" format SDBSF and changing the analysis
1 !                 context to non last track.
1 !
1 !     FALSE       If we could not (for one of several reasons) rebuild the
1 !                 the bad block files.
1 !
1 !--
2 BEGIN
2 LOCAL
2     return_status;
2
2 return_status = FALSE;                                  ! Assume we can't rebuild the bad block files.
2 IF .bad$gl_context [ctx_v_exercise]                     ! Check the context bits to see if we can
2 AND NOT .bad$gl_context [ctx_v_keep]                    ! attempt to rebuild them.
2 THEN
3     BEGIN
3     rebuild_dbsfs ();                                   ! We can try ...
3     return_status = TRUE;                               ! We have written them to the medium.
3     END
2 ELSE
3     BEGIN                                               ! Otherwise, if we aren't exercising the
3     IF NOT .bad$gl_context [ctx_v_exercise]             ! medium and no attempt will be made to
3     AND NOT .bad$gl_context [ctx_v_badblocks]           ! modify a non last track format SDBSF
3     THEN                                                ! if one is found, then go look for one.
3         IF get_nlt_sdbsf ()
3         THEN
4             BEGIN
4             bad$gl_context [ctx_v_ltdevice] = FALSE;    ! Found one, change to context.
4             return_status = TRUE;                       ! Set the status to allow us to continue.
3             END;
2     END;
2
2 RETURN .return_status;
2
1 END;     ! of ROUTINE recover_dbsfs
```

```
                                   000C 00000 RECOVER_DBSFS:
                                                      .WORD    Save R2,R3                          ; 0959
                 53      0000G  CF   9E 00002         MOVAB    BAD$GL_CONTEXT, R3                  ;
                                52   D4 00007         CLRL     RETURN_STATUS                       ; 0989
         0F      63             02   E1 00009         BBC      #2, BAD$GL_CONTEXT, 2$              ; 0990
         07      63             06   E0 0000D         BBS      #6, BAD$GL_CONTEXT, 1$              ; 0991
           0000V CF             00   FB 00011         CALLS    #0, REBUILD_DBSFS                   ; 0994
                 13                  11 00016         BRB      3$                                  ; 0995
         12      63             02   E0 00018 1$:     BBS      #2, BAD$GL_CONTEXT, 4$              ; 0999
                 0F             63   E8 0001C 2$:     BLBS     BAD$GL_CONTEXT, 4$                  ; 1000
           0000V CF             00   FB 0001F         CALLS    #0, GET_NLT_SDBSF                   ; 1002
                 07             50   E9 00024         BLBC     R0, 4$                              ;
              01 A3             01   8A 00027         BICB2    #1, BAD$GL_CONTEXT+1                ; 1005
                 52             01   D0 0002B 3$:     MOVL     #1, RETURN_STATUS                   ; 1006
                 50             52   D0 0002E 4$:     MOVL     RETURN_STATUS, R0                   ; 1010
                                     04 00031         RET                                          ; 1012

; Routine Size:  50 bytes,    Routine Base:  $CODE$ + 04AD

;  906          1013  1
```

```
1 %SBTTL  'rebuild_dbsfs -- Rebuild the MDBSF and SDBSF on last track devices'
1 ROUTINE rebuild_dbsfs : NOVALUE =
1 !++
1 !
1 ! Functional Description:
1 !
1 !     This procedure will rebuild the MDBSF and SDBSF on last track devices.
1 !     The context of the analysis MUST BE EXERCISE and NOKEEP.  This is to
1 !     assure that no data will be unintentionally lost due to the use of the
1 !     last track for the detected bad block files data.
1 !
1 ! Implicit Inputs:
1 !
1 !     Global Data.
1 !
1 ! Implicit Outputs:
1 !
1 !     The MDBSF and SDBSF will be recreated on the medium as empty files.
1 !
1 ! Side Effects:
1 !
1 !     If we did not manage to write the new MDBSF and SDBSF onto at least
1 !     half of the last track, we will issue a diagnostic to the user warning
1 !     him of the unreliable nature of the last track data.
1 !
1 !--
2 BEGIN
2 LOCAL
2     cnt : INITIAL (0);
2
2 bad$gl_func = IO$_WRITELBLK;                                  ! Set up for writes.
2 CH$FILL (-1, bad$k_page_size, .bad$ga_mdbsf);                 ! Clean the buffer out.
2 bad$ga_mdbsf [ltk_l_serial_number] = bad$gl_serialnum = 1; ! Default Serial Number.
2 bad$ga_mdbsf [ltk_w_unused] = 0;                              ! Clear unused field.
2 bad$ga_mdbsf [ltk_w_cartridge] = 0;                           ! Define it as a Data Cartridge.
2
2 INCR offset FROM 0 TO .bad$gl_sectors / .bad$gb_block_fact DO
3     IF NOT (bad$gl_status = bad$sync_io (.bad$ga_mdbsf,
3                                          bad$k_page_size,
3                                          .bad$gl_mdbsf_ptr + .offset))
2     THEN
2         cnt = .cnt + 1;
2
2 IF .cnt GEQ (.bad$gl_sectors / .bad$gb_block_fact) / 2  ! If we did not manage to write at
2 THEN                                                    ! least half of the last track,
2     SIGNAL (bad$_rebldwarn, 1, bad$ga_device);          ! send out a warning...
2
1 END;      ! of ROUTINE rebuild_dbsfs



                         00FC 00000 REBUILD_DBSFS:
                                     .WORD    Save R2,R3,R4,R5,R6,R7              ; 1015
                           57 D4 00002        CLRL     CNT                         ; 1040
     0000G  CF             20 D0 00004        MOVL     #32, BAD$GL_FUNC            ; 1044
```

```
                                      56     0000G  CF  D0 00009         MOVL    BAD$GA_MDBSF, R6                        ; 1045
   0200   8F        FF   8F           6E             00  2C 0000E         MOVC5   #0, (SP), #-1, #512, (R6)               ;
                                      66                    00016                                                          ;
                           0000G      CF             01  D0 00017         MOVL    #1, BAD$GL_SERIALNUM                    ; 1046
                                      66             01  7D 0001C         MOVQ    #1, (R6)                                ;
                                      53     0000G   CF  9A 0001F         MOVZBL  BAD$GB_BLOCK_FACT, R3                   ; 1050
                      53   0000G      CF             53  C7 00024         DIVL3   R3, BAD$GL_SECTORS, R3                  ;
                                      52             01  CE 0002A         MNEGL   #1, OFFSET                              ;
                                                     1F  11 0002D         BRB     2$                                      ;
                                           0000GDF42     9F 0002F  1$:    PUSHAB  @BAD$GL_MDBSF_PTR[OFFSET]               ; 1053
                                      7E     0200    8F  3C 00034         MOVZWL  #512, -(SP)                             ; 1051
                                             0000G   CF  DD 00039         PUSHL   BAD$GA_MDBSF                            ;
                      00000000G       00             03  FB 0003D         CALLS   #3, BAD$SYNC_IO                         ;
                           0000G      CF             50  D0 00044         MOVL    R0, BAD$GL_STATUS                       ;
                                      02             50  E8 00049         BLBS    R0, 2$                                  ;
                                                     57  D6 0004C         INCL    CNT                                     ; 1055
                      DD              52             53  F3 0004E  2$:    AOBLEQ  R3, OFFSET, 1$                          ; 1051
                                      50     0000G   CF  9A 00052         MOVZBL  BAD$GB_BLOCK_FACT, R0                   ; 1057
                      50   0000G      CF             50  C7 00057         DIVL3   R0, BAD$GL_SECTORS, R0                  ;
                                      50             02  C6 0005D         DIVL2   #2, R0                                  ;
                                      50             57  D1 00060         CMPL    CNT, R0                                 ;
                                                     13  19 00063         BLSS    3$                                      ;
                                             0000G   CF  9F 00065         PUSHAB  BAD$GA_DEVICE                           ; 1059
                                                     01  DD 00069         PUSHL   #1                                      ;
                                         00000000G   8F  DD 0006B         PUSHL   #BAD$_REBLDWARN                         ;
                      00000000G       00             03  FB 00071         CALLS   #3, LIB$SIGNAL                          ;
                                                     04  00078     3$:    RET                                             ; 1061

; Routine Size:  121 bytes,    Routine Base:  $CODE$ + 04DF


;  956          1062  1
```

```
1 %SBTTL  'get_sdbsf -- Get a valid copy of the SDBSF'
1 ROUTINE get_sdbsf : NOVALUE =
1 !++
1 !
1 ! Functional Description:
1 !
1 !    Obtain a valid copy of the Software Detected Bad Sector File.
1 !
1 ! Implicit Inputs:
1 !
1 !    bad$ga_mdbsf        the address of the MDBSF buffer.
1 !    bad$ga_sdbsf        the address of the SDBSF buffer.
1 !    bad$gl_mdbsf_ptr    the block number of the first block on the last track.
1 !    bad$gl_sdbsf_ptr    the address of the first free entry in the SDBSF buffer.
1 !
1 ! Side Effects:
1 !
1 !    If we are unable to read the SDBSF, if the SDBSF is FULL, or if it does
1 !    not correspond the MDBSF we will stop the analysis by issuing a FATAL
1 !    diagnostic.
1 !
1 !--
2 BEGIN
2 LOCAL
2     value_present;
2
2 value_present = FALSE;                                  ! The MDBSF was found and validated,
2                                                         ! now look for the SDBSF.
2 INCR block FROM 10 TO .bad$gl_sectors / .bad$gb_block_fact - 1 DO
3     BEGIN                                               ! Search the remainder of the last track
3     IF bad$gl_status = bad$sync_io (.bad$ga_sdbsf,     ! and validate the SDBSF's header.
3                                  bad$k_page_size,
3                                  .bad$gl_mdbsf_ptr + .block)
3     THEN
4         BEGIN
4         IF .bad$ga_sdbsf [ltk_l_serial_number] EQL      ! Validate the serial numbers.
4            .bad$ga_mdbsf [ltk_l_serial_number]
4         THEN
5             BEGIN
5             value_present = TRUE;                       ! They matched, the rest of the
5             EXITLOOP;                                   ! header should also be good.
4             END;
3         END;
2     END;
2
2 IF NOT .value_present
2 THEN
2     SIGNAL_STOP (bad$_sdbsfrfail, 1, bad$ga_device);
2
2 !
2 ! Point to the first free entry in the SDBSF.
2 !
2 value_present = FALSE;
2 INCR entry FROM .bad$ga_sdbsf TO .bad$ga_sdbsf + bad$k_page_size - 4 BY 4
2 DO
3     BEGIN
3     IF  ..entry EQL -1                                  ! An entry of -1 implies
```

```
1120 3       THEN                                                      ! the end of the bad block
1121 4           BEGIN                                                 ! information.
1122 4           value_present = TRUE;
1123 4           bad$gl_sdbsf_ptr = .entry;                            ! Set the address into
1124 4           EXITLOOP;                                             ! the global pointer and
1125 3           END;                                                  ! return.
1126 2       END;
1127 2
1128 2 IF NOT .value_present                                           ! If the bad block file
1129 2 THEN                                                            ! is full, issue a diagnostic
1130 2     SIGNAL_STOP (bad$_sdbsffull, 1, bad$ga_device);             ! and quit.
1131 2
1132 1 END;      ! of ROUTINE get_sdbsf


                                   007C 00000 GET_SDBSF:
                                                      .WORD    Save R2,R3,R4,R5,R6                           ; 1064
                56 00000000G  00    9E 00002          MOVAB    LIB$STOP, R6                                  ;
                55     0000G  CF    9E 00009          MOVAB    BAD$GA_SDBSF, R5                              ;
                54     0000G  CF    9A 0000E          MOVZBL   BAD$GB_BLOCK_FACT, R4                         ; 1091
     54       0000G  CF       54    C7 00013          DIVL3    R4, BAD$GL_SECTORS, R4                        ;
                     52       09    7D 00019          MOVQ     #9, BLOCK                                     ;
                              28    11 0001C          BRB      2$                                            ;
                   0000GDF42        9F 0001E  1$:     PUSHAB   @BAD$GL_MDBSF_PTR[BLOCK]                      ; 1095
                7E     0200   8F    3C 00023          MOVZWL   #512, -(SP)                                   ; 1093
                              65    DD 00028          PUSHL    BAD$GA_SDBSF                                  ;
          00000000G  00       03    FB 0002A          CALLS    #3, BAD$SYNC_IO                               ;
              0000G  CF       50    D0 00031          MOVL     R0, BAD$GL_STATUS                             ;
                     0D       50    E9 00036          BLBC     R0, 2$                                        ;
              0000G  DF  00   B5    D1 00039          CMPL     @BAD$GA_SDBSF, @BAD$GA_MDBSF                  ; 1099
                              05    12 0003F          BNEQ     2$                                            ;
                     53       01    D0 00041          MOVL     #1, VALUE_PRESENT                             ; 1102
                              04    11 00044          BRB      3$                                            ; 1101
     D4              52       54    F2 00046  2$:     AOBLSS   R4, BLOCK, 1$                                 ; 1091
                     0F       53    E8 0004A  3$:     BLBS     VALUE_PRESENT, 4$                             ; 1108
                    0000G     CF    9F 0004D          PUSHAB   BAD$GA_DEVICE                                 ; 1110
                              01    DD 00051          PUSHL    #1                                            ;
                000000000G    8F    DD 00053          PUSHL    #BAD$_SDBSFRFAIL                              ;
                     66       03    FB 00059          CALLS    #3, LIB$STOP                                  ;
                              53    D4 0005C  4$:     CLRL     VALUE_PRESENT                                 ; 1115
     51              65 000001FC 8F C1 0005E          ADDL3    #508, BAD$GA_SDBSF, R1                        ; 1116
                     50       65    D0 00066          MOVL     BAD$GA_SDBSF, ENTRY                           ;
                              16    11 00069          BRB      7$                                            ;
          FFFFFFFF   8F       60    D1 0006B  5$:     CMPL     (ENTRY), #-1                                  ; 1119
                              0A    12 00072          BNEQ     6$                                            ;
                     53       01    D0 00074          MOVL     #1, VALUE_PRESENT                             ; 1122
              0000G  CF       50    D0 00077          MOVL     ENTRY, BAD$GL_SDBSF_PTR                       ; 1123
                              08    11 0007C          BRB      8$                                            ; 1121
                     50       04    C0 0007E  6$:     ADDL2    #4, ENTRY                                     ; 1116
                     51       50    D1 00081  7$:     CMPL     ENTRY, R1                                     ;
                              E5    15 00084          BLEQ     5$                                            ;
                     0F       53    E8 00086  8$:     BLBS     VALUE_PRESENT, 9$                             ; 1128
                    0000G     CF    9F 00089          PUSHAB   BAD$GA_DEVICE                                 ; 1130
                              01    DD 0008D          PUSHL    #1                                            ;
```

```
                    00000000G  8F  DD 0008F        PUSHL   #BAD$_SDBSFFULL          ;
               66              03  FB 00095        CALLS   #3, LIB$STOP             ;
                                   04 00098 9$:    RET                              ; 1132

; Routine Size:  153 bytes,    Routine Base:  $CODE$ + 0558


; 1028          1133  1
```

```
1 %SBTTL  'get_nlt_sdbsf -- Get a non last track SDBSF'
1 ROUTINE get_nlt_sdbsf =
1 !++
1 !
1 ! Functional Description:
1 !
1 !     This is a non last track device, so search the last 256 blocks of the
1 !     device for the SDBSF.  Once its found validate it by checksumming (shared
1 !     routine from F11AACP) and verification of the fixed value fields of the
1 !     header.
1 !
1 ! Implicit Inputs:
1 !
1 !     bad$gl_maxblock      the total number of blocks on the device.
1 !     bad$ga_sdbsf         the address of the SDBSF buffer.
1 !
1 ! Side Effects:
1 !
1 !     If there is no SDBSF on the medium, an informational message will be issued
1 !     and a new SDBSF will be created.
1 !
1 !--
2 BEGIN
2 LOCAL
2     value_present;
2
2 value_present = FALSE;
2 DECR block FROM .bad$gl_maxblock TO .bad$gl_maxblock - 256 DO
3     BEGIN
3     bad$gl_status = bad$sync_io (.bad$ga_sdbsf,
3                                  bad$k_page_size,
3                                  .block);
3     IF .bad$gl_status
3     THEN
4         BEGIN
4         IF CHECKSUM2 (.bad$ga_sdbsf, bad$k_page_size - 2)      ! Sum of the entries, stored in the last wor
4         AND .bad$ga_sdbsf [nlt_b_cntfldsiz] EQL nlt_k_cntfldsiz ! Size of the count field in bytes.
4         AND .bad$ga_sdbsf [nlt_b_adrfldsiz] EQL nlt_k_adrfldsiz ! Size of the address field in bytes.
4         AND .bad$ga_sdbsf [nlt_b_usedbbdsc] LEQ nlt_k_available ! Number of used bad block descriptor
4         THEN                                                    ! (the count is in WORDs!)
5             BEGIN
5             value_present = TRUE;                               ! We found one, return.
5             EXITLOOP;
4             END;
3         END;
2     END;
2
2 IF NOT .value_present                                           ! If the SDBSF is not available, issue
2 THEN                                                            ! a diagnostic and create a new one.
3     BEGIN
3     SIGNAL (bad$_nobadinfo, 1, bad$ga_device);
3     init_sdbsf ();
2     END;
2
2 !
2 ! Point to the first free entry in the SDBSF before returning.
2 !
```

```
; 1087          1191  2 bad$gl_sdbsf_ptr = .bad$ga_sdbsf + nlt_k_headersiz +
; 1088          1192  2                     (.bad$ga_sdbsf [nlt_b_usedbbdsc] * 2);
; 1089          1193  2
; 1090          1194  2 RETURN .value_present;
; 1091          1195  2
; 1092          1196  1 END;      ! of ROUTINE get_nlt_sdbsf



                                          003C 00000 GET_NLT_SDBSF:
                                                            .WORD   Save R2,R3,R4,R5                              ; 1135
                   55       0000G   CF   9E 00002           MOVAB   BAD$GA_SDBSF, R5
                                    53   D4 00007           CLRL    VALUE_PRESENT                                 ; 1160
          54    0000G  CF 00000100  8F   C3 00009           SUBL3   #256, BAD$GL_MAXBLOCK, R4                     ; 1161
                   52       0000G   CF   D0 00013           MOVL    BAD$GL_MAXBLOCK, BLOCK                        ;
                                    4B   11 00018           BRB     3$                                            ;
                                    52   DD 0001A 1$:       PUSHL   BLOCK                                         ; 1165
                   7E       0200    8F   3C 0001C           MOVZWL  #512, -(SP)                                   ; 1163
                                    65   DD 00021           PUSHL   BAD$GA_SDBSF                                  ;
         00000000G 00               03   FB 00023           CALLS   #3, BAD$SYNC_IO                               ;
             0000G CF               50   D0 0002A           MOVL    R0, BAD$GL_STATUS                             ;
                   2F       0000G   CF   E9 0002F           BLBC    BAD$GL_STATUS, 2$                             ; 1166
                   7E       01FE    8F   3C 00034           MOVZWL  #510, -(SP)                                   ; 1169
                                    65   DD 00039           PUSHL   BAD$GA_SDBSF                                  ;
         00000000G 00               02   FB 0003B           CALLS   #2, CHECKSUM2                                 ;
                   1E               50   E9 00042           BLBC    R0, 2$                                        ;
                   01         00    B5   91 00045           CMPB    @BAD$GA_SDBSF, #1                             ; 1170
                   18               12      00049           BNEQ    2$                                            ;
                   50               65   D0 0004B           MOVL    BAD$GA_SDBSF, R0                              ; 1171
                   03         01    A0   91 0004E           CMPB    1(R0), #3                                     ;
                   0F               12      00052           BNEQ    2$                                            ;
                   50               65   D0 00054           MOVL    BAD$GA_SDBSF, R0                              ; 1172
              FC   8F         02    A0   91 00057           CMPB    2(R0), #252                                   ;
                   05               1A      0005C           BGTRU   2$                                            ;
                   53               01   D0 0005E           MOVL    #1, VALUE_PRESENT                             ; 1175
                   07               11      00061           BRB     4$                                            ; 1174
                                    52   D7 00063 2$:       DECL    BLOCK                                         ; 1161
                   54               52   D1 00065 3$:       CMPL    BLOCK, R4                                     ;
                                    B0   18 00068           BGEQ    1$                                            ;
                   18               53   E8 0006A 4$:       BLBS    VALUE_PRESENT, 5$                             ; 1181
                            0000G   CF   9F 0006D           PUSHAB  BAD$GA_DEVICE                                 ; 1184
                                    01   DD 00071           PUSHL   #1                                            ;
                        00000000G   8F   DD 00073           PUSHL   #BAD$_NOBADINFO                               ;
         00000000G 00               03   FB 00079           CALLS   #3, LIB$SIGNAL                                ;
             0000V CF               00   FB 00080           CALLS   #0, INIT_SDBSF                                ; 1185
                   51               65   D0 00085 5$:       MOVL    BAD$GA_SDBSF, R1                              ; 1191
                   50         02    A1   9A 00088           MOVZBL  2(R1), R0                                     ; 1192
             0000G CF         04  A140   3E 0008C           MOVAW   4(R1)[R0], BAD$GL_SDBSF_PTR                   ; 1191
                   50               53   D0 00093           MOVL    VALUE_PRESENT, R0                             ; 1194
                                         04 00096           RET                                                   ; 1196

; Routine Size:  151 bytes,    Routine Base:  $CODE$ + 05F1


; 1093          1197  1
```

```
1 %SBTTL  'init_sdbsf -- Create and initialize the header of the SDBSF'
1 ROUTINE init_sdbsf : NOVALUE =
1 !++
1 !
1 ! Functional Description:
1 !
1 !     This procedure will initialize the contents of a new SDBSF.  The pointer
1 !     to the first free entry will also be set.
1 !
1 ! Implicit Inputs:
1 !
1 !     bad$ga_sdbsf        the address of the SDBSF buffer.
1 !
1 ! Side Effects:
1 !
1 !     The header fields of the selected format bad block buffer are initialized
1 !     to the statically defined values.
1 !
1 !--
2 BEGIN
2
2 IF .bad$gl_context [ctx_v_ltdevice]
2 THEN
3     BEGIN         ! Last track device (copy static fields from the MDBSF).
3     bad$ga_sdbsf [ltk_l_serial_number] = .bad$ga_mdbsf [ltk_l_serial_number];
3     bad$ga_sdbsf [ltk_w_unused] = .bad$ga_mdbsf [ltk_w_unused];
3     bad$ga_sdbsf [ltk_w_cartridge] = .bad$ga_mdbsf [ltk_w_cartridge];
3     bad$gl_sdbsf_ptr = .bad$ga_sdbsf + ltk_k_headersiz;
3     END
2 ELSE
3     BEGIN         ! Non Last track device (use static defaults).
3     bad$ga_sdbsf [nlt_b_cntfldsiz] = nlt_k_cntfldsiz;   ! Fill in the count field size.
3     bad$ga_sdbsf [nlt_b_adrfldsiz] = nlt_k_adrfldsiz;   ! Fill in the address field size.
3     bad$ga_sdbsf [nlt_b_usedbbdsc] = 0;                 ! No descriptors used.
3     bad$ga_sdbsf [nlt_b_available] = nlt_k_available;   ! Total number of available descriptors.
3     bad$gl_sdbsf_ptr = .bad$ga_sdbsf + nlt_k_headersiz;
2     END;
2
1 END;     ! of ROUTINE init_sdbsf
```

```
                                   0000 00000 INIT_SDBSF:
                                                   .WORD   Save nothing                        ; 1199
           50      0000G   CF   D0 00002           MOVL    BAD$GA_SDBSF, R0                    ; 1222
           0F      0000G   CF   E9 00007           BLBC    BAD$GL_CONTEXT+1, 1$                ; 1219
           51      0000G   CF   D0 0000C           MOVL    BAD$GA_MDBSF, R1                    ; 1222
           60         61   7D 00011                MOVQ    (R1), (R0)
  0000G    CF         08   A0   9E 00014           MOVAB   8(R0), BAD$GL_SDBSF_PTR             ; 1225
                               04 0001A            RET                                         ; 1219
        60 FC000301   8F   D0 0001B 1$:            MOVL    #-67108095, (R0)                    ; 1229
  0000G    CF         04   A0   9E 00022           MOVAB   4(R0), BAD$GL_SDBSF_PTR             ; 1233
                               04 00028            RET                                         ; 1236

; Routine Size:  41 bytes,    Routine Base:  $CODE$ + 0688
```

```
; 1134        1237  1
```

```
; 1136    1238  1 %SBTTL  'get_pattern -- Process user supplied test pattern(s)'
; 1137    1239  1 ROUTINE get_pattern : NOVALUE =
; 1138    1240  1 !++
; 1139    1241  1 !
; 1140    1242  1 ! Functional Description:
; 1141    1243  1 !
; 1142    1244  1 !     Process the user supplied test pattern(s).
; 1143    1245  1 !
; 1144    1246  1 ! Implicit Inputs:
; 1145    1247  1 !
; 1146    1248  1 !     bad$gl_bad_term             the vector into which the pattern(s)
; 1147    1249  1 !                                 will be loaded.
; 1148    1250  1 !
; 1149    1251  1 !     bad$gb_term_count           the number of longwords which make
; 1150    1252  1 !                                 up the pattern.
; 1151    1253  1 !
; 1152    1254  1 ! Side Effects:
; 1153    1255  1 !
; 1154    1256  1 !     If the user neglects to supply at least one pattern, the context bit
; 1155    1257  1 !     (ctx_v_pattern) will be cleared, and the default pattern(s) will be used.
; 1156    1258  1 !
; 1157    1259  1 !--
; 1158    1260  2 BEGIN
; 1159    1261  2 bad$gl_context [ctx_v_pattern] = FALSE;                  ! Assume the default.
; 1160    1262  2 bad$gl_bad_term [0] = bad$gl_bad_term [1] = 0;           ! Clear the vector.
; 1161    1263  2 bad$gl_bad_term [2] = bad$gl_bad_term [3] = 0;           !
; 1162    1264  2 bad$gb_term_count = 0;                                   ! Clear the term count.
; 1163    1265  2
; 1164    1266  2 WHILE CLI$GET_VALUE (pattern, bad$ga_input_desc) DO      ! Process all supplied values.
; 1165    1267  3     BEGIN                                                !
; 1166    1268  3     bad$gl_context [ctx_v_pattern] = TRUE;               ! Flag that there are valid patterns available.
; 1167    1269  3     parse_value (number_tbl, numkey_tbl);                ! Parse the string into a value.
; 1168    1270  2     END;
; 1169    1271  2
; 1170    1272  1 END;      ! of ROUTINE get_pattern


                                    0000 00000 GET_PATTERN:
                                                       .WORD    Save nothing                          ; 1239
          0000G  CF            04  8A 00002            BICB2    #4, BAD$GL_CONTEXT+1                  ; 1261
                        0000G  CF  7C 00007            CLRQ     BAD$GL_BAD_TERM                       ; 1262
                        0000G  CF  7C 0000B            CLRQ     BAD$GL_BAD_TERM+8                     ; 1263
                        0000G  CF  94 0000F            CLRB     BAD$GB_TERM_COUNT                     ; 1264
                        0000G  CF  9F 00013 1$:        PUSHAB   BAD$GA_INPUT_DESC                     ; 1266
                        0000'  CF  9F 00017            PUSHAB   PATTERN                               ;
      00000000G  00            02  FB 0001B            CALLS    #2, CLI$GET_VALUE                     ;
                 14            50  E9 00022            BLBC     R0, 2$                                ;
          0000G  CF            04  88 00025            BISB2    #4, BAD$GL_CONTEXT+1                  ; 1268
                        0000'  CF  9F 0002A            PUSHAB   NUMKEY_TBL                            ; 1269
                        0000'  CF  9F 0002E            PUSHAB   NUMBER_TBL                            ;
          0000V  CF            02  FB 00032            CALLS    #2, PARSE_VALUE                       ;
                               DA  11 00037            BRB      1$                                    ; 1266
                                   04 00039 2$:        RET                                            ; 1272
```

```
; Routine Size:  58 bytes,    Routine Base:  $CODE$ + 06B1

; 1171          1273  1
```

```
1 %SBTTL  'process_badblocks -- Process user supplied bad block(s)'
1 ROUTINE process_badblocks =
1 !++
1 !
1 ! Functional Description:
1 !
1 !     Process the user supplied bad block information.  This is accomplished
1 !     by parsing the bad block(s) supplied by the user, validating them
1 !     against the device geometry, then if they are not already recorded in
1 !     a bad block file, they will be inserted.
1 !
1 ! Implicit Inputs:
1 !
1 !     bad$gl_bad_term              the vector containing the terms used to describe
1 !                                  the bad block(s).
1 !     bad$gb_term_count            the number of term used to describe the bad
1 !                                  block(s).
1 !
1 ! Side Effects:
1 !
1 !     If the block(s) specified are not already recorded in the bad block file(s)
1 !     they will be added.  If there are duplicate bad blocks specified, they will
1 !     cause a warning message to be displayed.
1 !
1 !     If we are prompting for the bad blocks, then if an invalid block number or
1 !     grammar error is encountered, we will issue a diagnostic, disregard the
1 !     last entry and prompt again.  If not prompting for bad blocks (they were
1 !     specified in the command), then we will issue a FATAL error diagnostic,
1 !     terminating the analysis.
1 !
1 !--
2 BEGIN
2 OWN
2     hibound,
2     lobound;
2
2 hibound = lobound = 0;                                    ! Assure we start out fresh.
2 bad$gb_term_count = 0;                                    ! Reset the count of terms
2 bad$gl_bad_term [0] = bad$gl_bad_term [1] = 0;            ! and each of the terms also.
2 bad$gl_bad_term [2] = bad$gl_bad_term [3] = 0;
2 parse_value (state_tbl, key_tbl);                         ! Parse the bad block info.
2
2 IF .bad$gb_term_count EQL 0                               ! Did we encounter a syntax error?
2 THEN                                                      ! Yes, so return for more input
2     RETURN FALSE;                                         ! (if any) to process.
2 !
2 ! Validate the address(es).  If validation fails, the validation routine
2 ! will depending on the processing context (interactive or command)
2 ! signal an error or halt execution of the image via a modification of
2 ! the message severity (set to STS$K_SEVERE).
2 !
2 IF .bad$gb_term_count GEQ 3                               ! Is the address in physical
2 THEN                                                      ! or logical address format?
3     BEGIN
3     IF NOT val_phy_adr (.bad$gl_bad_term [0],             ! Sector
3                         .bad$gl_bad_term [1],             ! Track
3                         .bad$gl_bad_term [2],             ! Cylinder
```

```
3                          .bad$gl_bad_term [3])          ! Count
3       THEN
3           RETURN FALSE;
3
3       lobound = bad$cvt_phy_log (.bad$gl_bad_term [0],      ! This is a valid address, form
3                                  .bad$gl_bad_term [1],      ! its equivalent logical block
3                                  .bad$gl_bad_term [2]);     ! number.
3
3       hibound = .lobound + .bad$gl_bad_term [3] - 1;
3       END
2   ELSE
3       BEGIN
3       IF NOT val_log_adr (.bad$gl_bad_term [0],             ! Logical Block Number
3                           .bad$gl_bad_term [1])             ! Count
3       THEN
3           RETURN FALSE;
3
3       lobound = .bad$gl_bad_term [0];                       ! Set the address bounds.
3       hibound = .lobound + .bad$gl_bad_term [1] - 1;
2       END;
2
2   IF .hibound LSS .lobound                                  ! Prevent mess up for one block,
2   THEN                                                      ! the hi must be same as low.
2       hibound = .lobound;
2   !
2   ! At this point we are satisfied with the addresses specified.  Determine
2   ! if the media is last track or non last track and look for a duplicate block.
2   ! If no duplicate is found, the block(s) will be entered in the SDBSF.
2   !
2   IF .bad$gl_context [ctx_v_ltdevice]
2   THEN
2       bad$check_ltk (.lobound, .hibound)
2   ELSE
2       bad$check_nlt (.lobound, .hibound);
2
2   RETURN TRUE;
2
1   END;        !of ROUTINE process_badblocks


                                        .PSECT  $OWN$,NOEXE,2

                          00000 HIBOUND:.BLKB   4
                          00004 LOBOUND:.BLKB   4


                                        .PSECT  $CODE$,NOWRT,2

                     000C 00000 PROCESS_BADBLOCKS:
                                        .WORD   Save R2,R3                      ; 1275
  53      0000G  CF  9E 00002           MOVAB   BAD$GL_BAD_TERM, R3             ;
  52      0000'  CF  9E 00007           MOVAB   HIBOUND, R2                     ;
                 62  7C 0000C           CLRQ    HIBOUND                         ; 1310
          0000G  CF  94 0000E           CLRB    BAD$GB_TERM_COUNT               ; 1311
                 63  7C 00012           CLRQ    BAD$GL_BAD_TERM                 ; 1312
```

```
                          08   A3   7C 00014        CLRQ    BAD$GL_BAD_TERM+8                       ; 1313
                       0000'   CF   9F 00017        PUSHAB  KEY_TBL                                 ; 1314
                       0000'   CF   9F 0001B        PUSHAB  STATE_TBL
            0000V  CF          02   FB 0001F        CALLS   #2, PARSE_VALUE
                   50   0000G  CF   9A 00024        MOVZBL  BAD$GB_TERM_COUNT, R0                   ; 1316
                               73   13 00029        BEQL    6$
                   03          50   91 0002B        CMPB    R0, #3                                  ; 1325
                               28   1F 0002E        BLSSU   1$
                   7E     08   A3   7D 00030        MOVQ    BAD$GL_BAD_TERM+8, -(SP)                ; 1330
                   7E          63   7D 00034        MOVQ    BAD$GL_BAD_TERM, -(SP)                  ; 1328
            0000V  CF          04   FB 00037        CALLS   #4, VAL_PHY_ADR
                   5F          50   E9 0003C        BLBC    R0, 6$
                   7E     04   A3   7D 0003F        MOVQ    BAD$GL_BAD_TERM+4, -(SP)                ; 1336
                               63   DD 00043        PUSHL   BAD$GL_BAD_TERM                         ; 1335
        00000000G  00          03   FB 00045        CALLS   #3, BAD$CVT_PHY_LOG
               04  A2          50   D0 0004C        MOVL    R0, LOBOUND
     50        04  A2     0C   A3   C1 00050        ADDL3   BAD$GL_BAD_TERM+12, LOBOUND, R0         ; 1339
                               15   11 00056        BRB     2$
                   7E          63   7D 00058 1$:    MOVQ    BAD$GL_BAD_TERM, -(SP)                  ; 1343
            0000V  CF          02   FB 0005B        CALLS   #2, VAL_LOG_ADR
                   3B          50   E9 00060        BLBC    R0, 6$
               04  A2          63   D0 00063        MOVL    BAD$GL_BAD_TERM, LOBOUND                ; 1348
     50        04  A2     04   A3   C1 00067        ADDL3   BAD$GL_BAD_TERM+4, LOBOUND, R0          ; 1349
                   62     FF   A0   9E 0006D 2$:    MOVAB   -1(R0), HIBOUND
                   50     04   A2   D0 00071        MOVL    LOBOUND, R0                             ; 1352
                   50          62   D1 00075        CMPL    HIBOUND, R0
                               03   18 00078        BGEQ    3$
                   62          50   D0 0007A        MOVL    R0, HIBOUND                             ; 1354
                   0D   0000G  CF   E9 0007D 3$:    BLBC    BAD$GL_CONTEXT+1, 4$                    ; 1360
                               62   DD 00082        PUSHL   HIBOUND                                 ; 1362
                               50   DD 00084        PUSHL   R0
        00000000G  00          02   FB 00086        CALLS   #2, BAD$CHECK_LTK
                               0B   11 0008D        BRB     5$
                               62   DD 0008F 4$:    PUSHL   HIBOUND                                 ; 1364
                               50   DD 00091        PUSHL   R0
        00000000G  00          02   FB 00093        CALLS   #2, BAD$CHECK_NLT
                   50          01   D0 0009A 5$:    MOVL    #1, R0                                  ; 1366
                                    04 0009D        RET
                               50   D4 0009E 6$:    CLRL    R0                                      ; 1368
                                    04 000A0        RET

; Routine Size:  161 bytes,    Routine Base:  $CODE$ + 06EB


; 1268          1369  1
```

```
1 %SBTTL  'parse_value -- Parse and process user supplied value(s)'
1 ROUTINE parse_value (state, key) : NOVALUE =
1 !++
1 !
1 ! Functional Description:
1 !
1 !     This routine is responsible for parsing user supplied values which
1 !     are used by /BAD_BLOCKS, /EXERCISE=PATTERN, etc..
1 !
1 ! Inputs:
1 !
1 !     state       adr     address of the state table to use.
1 !     key         adr     address of the key table to use.
1 !
1 ! Implicit Inputs:
1 !
1 !     The address of the TPARSE control block.
1 !
1 !     The descriptor pointer to the string to be parsed (bad$ga_input_desc).
1 !
1 ! Implicit Outputs:
1 !
1 !     The information vector bad$gl_bad_term [x] will contain the user
1 !     supplied data values.
1 !
1 !--
2 BEGIN
2 LOCAL
2     tparse_blk   : BLOCK [TPA$K_LENGTH0, BYTE];          ! TPARSE control block.
2
2 IF bad$str_trim NEQ 0                                    ! Trim the input line if we are stand alone
2 THEN    bad$str_trim (bad$ga_input_desc);                ! to preclude erroneous errors during the parse.
2
2 tparse_blk [tpa$l_count] = TPA$K_COUNT0;                 ! Initial longword count.
2 tparse_blk [tpa$l_options] = TPA$M_ABBREV;               ! Allow abbreviations.
2 tparse_blk [tpa$l_stringcnt] = .bad$ga_input_desc [dsc$w_length];
2 tparse_blk [tpa$l_stringptr] = .bad$ga_input_desc [dsc$a_pointer];
2
2 LIB$TPARSE (tparse_blk, .state, .key);                   ! Parse the value.
2
2 IF bad$str_trim NEQ 0                                    ! Restore the buffer size if we are stand alone
2 THEN    bad$ga_input_desc [dsc$w_length] = bad$k_input_len;
2
1 END;    ! of ROUTINE parse_value
```

```
                              000C 00000 PARSE_VALUE:
                                         .WORD   Save R2,R3                       ; 1371
53         0000G   CF   9E 00002         MOVAB   BAD$GA_INPUT_DESC, R3            ;
5E                 24   C2 00007         SUBL2   #36, SP                          ;
50         0000G   CF   9E 0000A         MOVAB   BAD$STR_TRIM, R0                 ; 1400
                   52   D4 0000F         CLRL    R2                               ;
                   50   D5 00011         TSTL    R0                               ;
                   09   13 00013         BEQL    1$                               ;
```

```
                                    52  D6 00015          INCL    R2                                          ;
                                    53  DD 00017          PUSHL   R3                                          ; 1401
                  0000G  CF         01  FB 00019          CALLS   #1, BAD$STR_TRIM                            ;
                         6E         08  D0 0001E 1$:      MOVL    #8, TPARSE_BLK                              ; 1403
                   04    AE         02  D0 00021          MOVL    #2, TPARSE_BLK+4                            ; 1404
                   08    AE         63  3C 00025          MOVZWL  BAD$GA_INPUT_DESC, TPARSE_BLK+8             ; 1405
                   0C    AE    04   A3  D0 00029          MOVL    BAD$GA_INPUT_DESC+4, TPARSE_BLK+12          ; 1406
                         7E    04   AC  7D 0002E          MOVQ    STATE, -(SP)                                ; 1408
                               08   AE  9F 00032          PUSHAB  TPARSE_BLK                                  ;
              00000000G  00         03  FB 00035          CALLS   #3, LIB$TPARSE                              ;
                         04         52  E9 0003C          BLBC    R2, 2$                                      ; 1410
                         63    84   8F  9B 0003F          MOVZBW  #132, BAD$GA_INPUT_DESC                     ; 1411
                                        04 00043 2$:      RET                                                 ; 1413

; Routine Size:  68 bytes,    Routine Base:  $CODE$ + 078C


; 1314          1414  1
```

```
; 1316        1415 1 %SBTTL 'TPARSE action routines'
; 1317        1416 1 ROUTINE count_term =
; 1318        1417 1 !++
; 1319        1418 1 !
; 1320        1419 1 ! Functional Description:
; 1321        1420 1 !
; 1322        1421 1 !     This routine is called by TPARSE to indicate the number
; 1323        1422 1 !     of terms which describe the user supplied bad block(s).
; 1324        1423 1 !
; 1325        1424 1 ! Implicit Outputs:
; 1326        1425 1 !
; 1327        1426 1 !     bad$gl_bad_term              The appropriate longword in this vector is
; 1328        1427 1 !                                  loaded with the current value of the parse.
; 1329        1428 1 !
; 1330        1429 1 !     bad$gb_term_count            The count value is used after parsing the
; 1331        1430 1 !                                  current term to describe the form of bad block
; 1332        1431 1 !                                  information specified.
; 1333        1432 1 !
; 1334        1433 1 !                                  1 => lbn
; 1335        1434 1 !                                  2 => lbn:count
; 1336        1435 1 !                                  3 => sector.track.cylinder
; 1337        1436 1 !                                  4 => sector.track.cylinder:count
; 1338        1437 1 !
; 1339        1438 1 !--
; 1340        1439 2 BEGIN
; 1341        1440 2
; 1342        1441 2 def_tparse_args;
; 1343        1442 2
; 1344        1443 2 bad$gl_bad_term [.bad$gb_term_count] = .tparse_args [TPA$L_NUMBER];
; 1345        1444 2 bad$gb_term_count = .bad$gb_term_count + 1;
; 1346        1445 2
; 1347        1446 2 IF .bad$gb_term_count GTR 4
; 1348        1447 2 THEN RETURN FALSE;
; 1349        1448 2
; 1350        1449 2 RETURN TRUE;
; 1351        1450 2
; 1352        1451 1 END;     ! of ROUTINE count_term


                                      0004 00000 COUNT_TERM:
                                                         .WORD    Save R2                                  ; 1416
             52      0000G  CF  9E 00002                 MOVAB    BAD$GB_TERM_COUNT, R2
             50             62  9A 00007                 MOVZBL   BAD$GB_TERM_COUNT, R0                    ; 1443
      0000GCF40         1C  AC  D0 0000A                 MOVL     28(TPARSE_ARGS), BAD$GL_BAD_TERM[R0]
                            62  96 00011                 INCB     BAD$GB_TERM_COUNT                        ; 1444
             04             62  91 00013                 CMPB     BAD$GB_TERM_COUNT, #4                    ; 1446
                            04  1A 00016                 BGTRU    1$
             50             01  D0 00018                 MOVL     #1, R0                                   ; 1449
                                04 0001B                 RET
                        50  D4 0001C  1$:                CLRL     R0                                       ; 1451
                                04 0001E                 RET

; Routine Size:  31 bytes,    Routine Base:  $CODE$ + 07D0
```

```
; 1353          1452  1
```

```
; 1355      1453  1 ROUTINE syntax_err =
; 1356      1454  1 !++
; 1357      1455  1 !
; 1358      1456  1 ! Functional Description:
; 1359      1457  1 !
; 1360      1458  1 !     This routine issues a diagnotic to inform the user of an invalid
; 1361      1459  1 !     argument in the bad block specification.  The term count is reset
; 1362      1460  1 !     to indicate an invalid bad block specification was encountered.
; 1363      1461  1 !
; 1364      1462  1 ! Implicit Outputs:
; 1365      1463  1 !
; 1366      1464  1 !     bad$gb_term_count           is reset, to indicate no value present.
; 1367      1465  1 !
; 1368      1466  1 ! Side Effects:
; 1369      1467  1 !
; 1370      1468  1 !     Since there was a missing argument in this block spec, it constitutes
; 1371      1469  1 !     an unrecoverable error.  If we are prompting for bad blocks, issue the
; 1372      1470  1 !     diagnostic and prompt for the next block.  If we are processing bad
; 1373      1471  1 !     blocks issued from the command line, then terminate the analysis after
; 1374      1472  1 !     issuing the diagnostic.
; 1375      1473  1 !
; 1376      1474  1 !--
; 1377      1475  2 BEGIN
; 1378      1476  2 bad$gb_term_count = 0;
; 1379      1477  2
; 1380      1478  2 SIGNAL (IF .bad$gl_context [ctx_v_interactive]
; 1381      1479  2         THEN bad$_ivblkent
; 1382      1480  2         ELSE bad$_ivblkent + STS$K_SEVERE,
; 1383      1481  2         1, bad$ga_input_desc);
; 1384      1482  2
; 1385      1483  2 RETURN  TRUE;
; 1386      1484  2
; 1387      1485  1 END;     ! of ROUTINE syntax_err


                                       0000 00000 SYNTAX_ERR:
                                                          .WORD   Save nothing                          ; 1453
                    0000G  CF   94 00002                  CLRB    BAD$GB_TERM_COUNT                     ; 1476
                    0000G  CF   9F 00006                  PUSHAB  BAD$GA_INPUT_DESC                     ; 1478
                           01   DD 0000A                  PUSHL   #1                                    ;
      08     0000G  CF     05   E1 0000C                  BBC     #5, BAD$GL_CONTEXT, 1$                ;
                00000000G  8F   DD 00012                  PUSHL   #BAD$_IVBLKENT                        ;
                           06   11 00018                  BRB     2$                                    ;
                00000000G  8F   DD 0001A 1$:              PUSHL   #BAD$_IVBLKENT+4                      ; 1480
         00000000G  00     03   FB 00020 2$:              CALLS   #3, LIB$SIGNAL                        ; 1478
                    50     01   D0 00027                  MOVL    #1, R0                                ; 1483
                                04 0002A                  RET                                           ; 1485

; Routine Size:  43 bytes,    Routine Base:  $CODE$ + 07EF


; 1388      1486  1
```

```
; 1390          1487  1 ROUTINE bad_value =
; 1391          1488  1 !++
; 1392          1489  1 !
; 1393          1490  1 ! Functional Description:
; 1394          1491  1 !
; 1395          1492  1 !     This routine issues a diagnotic to inform the user of an invalid
; 1396          1493  1 !     value on the PATTERN keyword.
; 1397          1494  1 !
; 1398          1495  1 ! Side Effects:
; 1399          1496  1 !
; 1400          1497  1 !     Since there was an invalid pattern value, it constitutes an unrecoverable
; 1401          1498  1 !     error.  Since we are processing the value from the command line, terminate
; 1402          1499  1 !     the analysis after issuing the diagnostic.
; 1403          1500  1 !
; 1404          1501  1 !--
; 1405          1502  2 BEGIN
; 1406          1503  2
; 1407          1504  2 SIGNAL (bad$_badvalue, 1, bad$ga_input_desc);
; 1408          1505  2 RETURN TRUE;
; 1409          1506  2
; 1410          1507  1 END;      ! of ROUTINE bad_value



                                        0000 00000 BAD_VALUE:
                                                           .WORD   Save nothing                         ; 1487
                            0000G  CF   9F 00002           PUSHAB  BAD$GA_INPUT_DESC                    ; 1504
                                   01   DD 00006           PUSHL   #1                                   ;
                        00000000G  8F   DD 00008           PUSHL   #BAD$_BADVALUE                       ;
            00000000G  00          03   FB 0000E           CALLS   #3, LIB$SIGNAL                       ;
                       50          01   D0 00015           MOVL    #1, R0                               ; 1505
                                        04 00018           RET                                          ; 1507

; Routine Size:  25 bytes,    Routine Base:  $CODE$ + 081A


; 1411          1508  1
```

```
1 %SBTTL  'val_phy_adr -- Validate Physical Address'
1 ROUTINE val_phy_adr (sec, trk, cyl, cnt) =
1 !++
1 !
1 ! Functional Description:
1 !
1 !     Validate Physical Address Range
1 !
1 ! Inputs:
1 !
1 !     sec         val     the sector number supplied by the user.
1 !     trk         val     the track number supplied by the user.
1 !     cyl         val     the cylinder number supplied by the user.
1 !     cnt         val     the count of contigous sectors to mark bad.
1 !
1 ! Implicit Inputs:
1 !
1 !     bad$gl_cylinders   the number of cylinders on the device.
1 !     bad$gl_maxblock    the total number of blocks on the device.
1 !     bad$gl_sectors     the number of sectors per track on the device.
1 !     bad$gl_tracks      the number of tracks per cylinder on the device.
1 !
1 ! Side Effects:
1 !
1 !     If any of the parts which constitute the physical address are not
1 !     valid, we will issue a diagnostic and select on of the following actions.
1 !
1 !         If prompting, disregard the last entry and prompt for more input.
1 !
1 !         If processing blocks from the command line, the severity will be set
1 !         to FATAL, resulting in termination of the analysis.
1 !
1 !--
2 BEGIN
2 OWN
2     hibound,
2     status;
2
2 status = TRUE;                                              ! Assume address is ok.
2
2 IF .sec GTR .bad$gl_sectors - 1                             ! Is the sector number
2 THEN                                                        ! within range?
2     status = addr_error (sector, .sec);
2
2 IF .trk GTR .bad$gl_tracks - 1                              ! Is the track number
2 THEN                                                        ! within range?
2     status = addr_error (track, .trk);
2
2 IF .cyl GTR .bad$gl_cylinders - 1                           ! Is the cylinder number
2 THEN                                                        ! within range?
2     status = addr_error (cylinder, .cyl);
2
2 IF .cnt NEQ 0                                               ! Should we consider the count field?
2 THEN
3     BEGIN                                                   ! Yes.
3     hibound = bad$cvt_phy_log(.sec, .trk, .cyl);            ! Convert to logical address.
3     IF .hibound + .cnt GTR .bad$gl_maxblock - 1             ! Does the COUNT field
```

```
; 1470      1566 3       THEN                                       ! push us over the end
; 1471      1567 3           status = addr_error (count, .cnt);     ! of the media?
; 1472      1568 2       END;                                       ! Inform user of the
; 1473      1569 2                                                  ! invalid address generated.
; 1474      1570 2   RETURN .status;
; 1475      1571 2
; 1476      1572 1   END;        ! of ROUTINE val_phy_adr


                                                        .PSECT  $OWN$,NOEXE,2

                                  00008 HIBOUND:.BLKB   4
                                  0000C STATUS: .BLKB   4


                                                        .PSECT  $CODE$,NOWRT,2

                             001C 00000 VAL_PHY_ADR:
                                                        .WORD   Save R2,R3,R4                           ; 1510
              54     0000V  CF  9E 00002                MOVAB   ADDR_ERROR, R4                          ;
              53     0000'  CF  9E 00007                MOVAB   STATUS, R3                              ;
              63            01  D0 0000C                MOVL    #1, STATUS                              ; 1547
  50   0000G  CF            01  C3 0000F                SUBL3   #1, BAD$GL_SECTORS, R0                  ; 1549
              50       04   AC  D1 00015                CMPL    SEC, R0                                 ;
                            0D  15 00019                BLEQ    1$                                      ;
                       04   AC  DD 0001B                PUSHL   SEC                                     ; 1551
                     0000'  CF  9F 0001E                PUSHAB  SECTOR                                  ;
              64            02  FB 00022                CALLS   #2, ADDR_ERROR                          ;
              63            50  D0 00025                MOVL    R0, STATUS                              ;
  50   0000G  CF            01  C3 00028 1$:            SUBL3   #1, BAD$GL_TRACKS, R0                   ; 1553
              50       08   AC  D1 0002E                CMPL    TRK, R0                                 ;
                            0D  15 00032                BLEQ    2$                                      ;
                       08   AC  DD 00034                PUSHL   TRK                                     ; 1555
                     0000'  CF  9F 00037                PUSHAB  TRACK                                   ;
              64            02  FB 0003B                CALLS   #2, ADDR_ERROR                          ;
              63            50  D0 0003E                MOVL    R0, STATUS                              ;
  50   0000G  CF            01  C3 00041 2$:            SUBL3   #1, BAD$GL_CYLINDERS, R0                ; 1557
              50       0C   AC  D1 00047                CMPL    CYL, R0                                 ;
                            0D  15 0004B                BLEQ    3$                                      ;
                       0C   AC  DD 0004D                PUSHL   CYL                                     ; 1559
                     0000'  CF  9F 00050                PUSHAB  CYLINDER                                ;
              64            02  FB 00054                CALLS   #2, ADDR_ERROR                          ;
              63            50  D0 00057                MOVL    R0, STATUS                              ;
              52       10   AC  D0 0005A 3$:            MOVL    CNT, R2                                 ; 1561
                            2E  13 0005E                BEQL    4$                                      ;
              7E       08   AC  7D 00060                MOVQ    TRK, -(SP)                              ; 1564
              04            AC  DD 00064                PUSHL   SEC                                     ;
     00000000G  00          03  FB 00067                CALLS   #3, BAD$CVT_PHY_LOG                     ;
            FC  A3          50  D0 0006E                MOVL    R0, HIBOUND                             ;
  51          52       FC   A3  C1 00072                ADDL3   HIBOUND, R2, R1                         ; 1565
  50   0000G  CF            01  C3 00077                SUBL3   #1, BAD$GL_MAXBLOCK, R0                 ;
              50            51  D1 0007D                CMPL    R1, R0                                  ;
                            0C  15 00080                BLEQ    4$                                      ;
                            52  DD 00082                PUSHL   R2                                      ; 1567
                     0000'  CF  9F 00084                PUSHAB  COUNT                                   ;
```

```
                     64          02  FB 00088       CALLS   #2, ADDR_ERROR                       ;
                     63          50  D0 0008B       MOVL    R0, STATUS                           ;
                     50          63  D0 0008E 4$:   MOVL    STATUS, R0                           ; 1570
                                     04 00091       RET                                          ; 1572

; Routine Size:  146 bytes,    Routine Base:  $CODE$ + 0833


; 1477          1573  1
```

```
1 %SBTTL  'val_log_adr -- Validate Logical Address'
1 ROUTINE val_log_adr (lbn, cnt) =
1 !++
1 !
1 ! Functional Description:
1 !
1 !    Validate Logical Block Number Address range.
1 !
1 ! Inputs:
1 !
1 !    lbn         val     the block number to validate.
1 !    cnt         val     the number of contiguous block to validate.
1 !
1 ! Implicit Inputs:
1 !
1 !    bad$gl_maxblock     the total number of blocks on the device.
1 !
1 ! Side Effects:
1 !
1 !    If any of the logical block number or the count force us to generate an
1 !    invalid address, we will issue a diagnostic and select on of the following
1 !    actions.
1 !
1 !        If prompting, disregard the last entry and prompt for more input.
1 !
1 !        If processing blocks from the command line, the severity will be set
1 !        to FATAL, resulting in termination of the analysis.
1 !
1 !--
2 BEGIN
2 LOCAL
2     status;
2
2 status = TRUE;                                          ! Assume address is ok.
2
2 IF .lbn GTR .bad$gl_maxblock - 1                        ! Is this a valid lbn?
2 THEN
2     status = addr_error (logical_blk, .lbn);
2
2 IF .cnt NEQ 0                                           ! Should we consider the count?
2 THEN                                                    ! Yes.
2     IF .lbn + .cnt GTR .bad$gl_maxblock - 1             ! Does the count push
2     THEN                                                ! us off the end of the
2         status = addr_error (count, .cnt);              ! media.
2
2 RETURN .status;
2
1 END;     ! of ROUTINE val_log_adr


                                        000C 00000 VAL_LOG_ADR:
                                                   .WORD    Save R2,R3                          ; 1575
                   50             01 D0 00002      MOVL     #1, STATUS                          ; 1607
     51      0000G CF             01 C3 00005      SUBL3    #1, BAD$GL_MAXBLOCK, R1             ; 1609
```

```
                    51      04  AC  D1 0000B      CMPL    LBN, R1                        ;
                                0C  15 0000F      BLEQ    1$                             ;
                            04  AC  DD 00011      PUSHL   LBN                            ; 1611
                         0000'  CF  9F 00014      PUSHAB  LOGICAL_BLK                    ;
               0000V  CF        02  FB 00018      CALLS   #2, ADDR_ERROR                 ;
                    52      08  AC  D0 0001D 1$:  MOVL    CNT, R2                        ; 1613
                                1B  13 00021      BEQL    2$                             ;
          53        52      04  AC  C1 00023      ADDL3   LBN, R2, R3                    ; 1615
          51   0000G  CF        01  C3 00028      SUBL3   #1, BAD$GL_MAXBLOCK, R1        ;
                    51          53  D1 0002E      CMPL    R3, R1                         ;
                                0B  15 00031      BLEQ    2$                             ;
                                52  DD 00033      PUSHL   R2                             ; 1617
                         0000'  CF  9F 00035      PUSHAB  COUNT                          ;
               0000V  CF        02  FB 00039      CALLS   #2, ADDR_ERROR                 ;
                                04 0003E 2$:      RET                                    ; 1621

; Routine Size:  63 bytes,    Routine Base:  $CODE$ + 08C5

; 1527          1622  1
```

```
; 1529    1623  1 %SBTTL  'addr_error -- Address error handler'
; 1530    1624  1 ROUTINE addr_error (desc, val) =
; 1531    1625  1 !+
; 1532    1626  1 !
; 1533    1627  1 ! Functional Description:
; 1534    1628  1 !
; 1535    1629  1 !     There is an error in the address specification, inform the user of the
; 1536    1630  1 !     problem area.
; 1537    1631  1 !
; 1538    1632  1 ! Side Effects:
; 1539    1633  1 !
; 1540    1634  1 !     If prompting, disregard the last entry and prompt for more input.
; 1541    1635  1 !
; 1542    1636  1 !     If processing blocks from the command line, the severity will be set
; 1543    1637  1 !     to FATAL, resulting in termination of the analysis.
; 1544    1638  1 !
; 1545    1639  1 !--
; 1546    1640  2 BEGIN
; 1547    1641  2 bad$gb_term_count = 0;                                 ! Reset the term count.
; 1548    1642  2
; 1549    1643  2 If bad$str_trim NEQ 0                                  ! If we are stand alone, trim the string
; 1550    1644  2 THEN    bad$str_trim (bad$ga_input_desc);              ! of trailing blanks, tabs and carriage return.
; 1551    1645  2
; 1552    1646  2 SIGNAL (IF .bad$gl_context [ctx_v_interactive]         ! If the user is interactively entering
; 1553    1647  2         THEN bad$_ivblknum                             ! bad blocks, reject this entry.
; 1554    1648  2         ELSE bad$_ivblknum + STS$K_SEVERE,             ! Otherwise, stop the analysis here.
; 1555    1649  2         3,                                             ! Three pieces of information...
; 1556    1650  2         bad$ga_input_desc,                             ! The current input spec,
; 1557    1651  2         .desc,                                         ! the part of the bad block spec out of range,
; 1558    1652  2         .val);                                         ! and the offending value.
; 1559    1653  2
; 1560    1654  2 IF bad$str_trim NEQ 0                                  ! Restore the buffer size if we are stand alone.
; 1561    1655  2 THEN    bad$ga_input_desc [dsc$w_length] = bad$k_input_len;
; 1562    1656  2
; 1563    1657  2 RETURN FALSE;
; 1564    1658  2
; 1565    1659  1 END;    ! of ROUTINE addr_error


                                        000C 00000 ADDR_ERROR:
                                                        .WORD   Save R2,R3                          ; 1624
                  53      0000G  CF  9E 00002           MOVAB   BAD$GA_INPUT_DESC, R3               ;
                          0000G  CF  94 00007           CLRB    BAD$GB_TERM_COUNT                   ; 1641
                  50      0000G  CF  9E 0000B           MOVAB   BAD$STR_TRIM, R0                    ; 1643
                                 52  D4 00010           CLRL    R2                                  ;
                                 50  D5 00012           TSTL    R0                                  ;
                                 09  13 00014           BEQL    1$                                  ;
                                 52  D6 00016           INCL    R2                                  ;
                                 53  DD 00018           PUSHL   R3                                  ; 1644
               0000G  CF         01  FB 0001A           CALLS   #1, BAD$STR_TRIM                    ;
                      7E     04  AC  7D 0001F  1$:      MOVQ    DESC, -(SP)                         ; 1651
                                 53  DD 00023           PUSHL   R3                                  ; 1646
                                 03  DD 00025           PUSHL   #3                                  ;
          08   0000G  CF         05  E1 00027           BBC     #5, BAD$GL_CONTEXT, 2$              ;
```

```
                          00000000G  8F  DD 0002D        PUSHL   #BAD$_IVBLKNUM                          ;
                                     06  11 00033        BRB     3$                                      ;
                          00000000G  8F  DD 00035 2$:    PUSHL   #BAD$_IVBLKNUM+4                        ; 1648
              00000000G  00          05  FB 0003B 3$:    CALLS   #5, LIB$SIGNAL                          ; 1646
                         04          52  E9 00042        BLBC    R2, 4$                                  ; 1654
                         63     84   8F  9B 00045        MOVZBW  #132, BAD$GA_INPUT_DESC                 ; 1655
                                     50  D4 00049 4$:    CLRL    R0                                      ; 1657
                                         04 0004B        RET                                             ; 1659

; Routine Size:  76 bytes,    Routine Base:  $CODE$ + 0904


; 1566          1660  1
; 1567          1661  1 END      ! of MODULE badinit
; 1568          1662  0 ELUDOM




                                                                 .EXTRN  LIB$SIGNAL, LIB$STOP

;                                      PSECT SUMMARY
;
;       Name                      Bytes                       Attributes
;
;    $PLIT$                         340  NOVEC,NOWRT,  RD ,NOEXE,NOSHR,  LCL,  REL,  CON,NOPIC,ALIGN(2)
;    _LIB$KEY0$                       0  NOVEC,NOWRT,  RD ,  EXE,  SHR,  LCL,  REL,  CON,  PIC,ALIGN(1)
;    _LIB$STATE$                    194  NOVEC,NOWRT,  RD ,  EXE,  SHR,  LCL,  REL,  CON,  PIC,ALIGN(1)
;    $CODE$                        2384  NOVEC,NOWRT,  RD ,  EXE,NOSHR,  LCL,  REL,  CON,NOPIC,ALIGN(2)
;    $OWN$                           16  NOVEC,  WRT,  RD ,NOEXE,NOSHR,  LCL,  REL,  CON,NOPIC,ALIGN(2)



;                                 Library Statistics
;
;                                            -------- Symbols --------      Pages       Processing
;       File                                 Total    Loaded   Percent      Mapped      Time
;
;    _$255$DUA28:[SYSLIB]LIB.L32;1           18619        29         0       1000         00:01.5
;    _$255$DUA28:[SYSLIB]TPAMAC.L32;1           42        27        64         14         00:00.1




;                                      COMMAND QUALIFIERS

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

; Size:           2384 code + 550 data bytes
; Run Time:          00:38.7
; Elapsed Time:      02:55.8
; Lines/CPU Min:       2579
; Lexemes/CPU-Min: 57142
```

```
; Memory Used:  326 pages
; Compilation Complete
```

0017 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

BADBLOCKS
LIS

BAD

STABAD
MAP

BADDATA
LIS

ANALYZBAD
MAP

ANALYZCMD
CLD

BADDEF
SDL

WRITESAVE
LIS

BADINIT
LIS

ANALYZCMD
LIS

0018 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

BADREPORT
LIS

BADBLK

SCANFILE
LIS

BADBLOCK
MAP

BADMAIN
LIS

BADIO
LIS

GETREQ
LIS

BADMSG
LIS

BADSTASYS
LIS