OpenVMS Bad Block Locator Utility Manual

Order Number: AA–PS69A–TE

May 1993

This document describes how to use the Bad Block Locator Utility on VMS operating systems.

Revision/Update Information: This manual supersedes the *VMS Bad Block Locator Utility Manual*, Version 5.2

Software Version: OpenVMS AXP Version 1.5
OpenVMS VAX Version 6.0

Digital Equipment Corporation
Maynard, Massachusetts

May 1993

The postpaid Reader's Comments forms at the end of this document request your critical evaluation to assist in preparing future documentation.

ZK4535

This document was prepared using VAX DOCUMENT, Version 2.1.

Contents

Preface

Intended Audience

This manual is intended for VMS system managers, operators, and system programmers.

Document Structure

This document consists of the following three sections:

- Description—Provides a full description of the Bad Block Locator Utility (BAD).
- Usage Summary—Outlines the following BAD information:
  - –Invoking the utility
  - –Exiting from the utility
  - –Directing output
  - –Restrictions or privileges required
- Qualifiers—Describes BAD qualifiers, including format, parameters, and examples.

Associated Documents

For additional information on the topics covered in this document, see the *OpenVMS DCL Dictionary* and the *OpenVMS System Manager's Manual*. Also see the *OpenVMS System Messages and Recovery Procedures Reference Manual* for explanations of diagnostic messages and suggestions for user action.

Conventions

In this manual, every use of VMS means both the OpenVMS AXP and the OpenVMS VAX operating system.

The following conventions are also used in this manual:

Ctrl/*x*	A sequence such as Ctrl/*x* indicates that you must hold down the key labeled Ctrl while you press another key or a pointing device button.
Return	In examples, a key name enclosed in a box indicates that you press a key on the keyboard. (In text, a key name is not enclosed in a box.)

. . .	A horizontal ellipsis in examples indicates one of the following possibilities: • Additional optional arguments in a statement have been omitted. • The preceding item or items can be repeated one or more times. • Additional parameters, values, or other information can be entered.
. . .	A vertical ellipsis indicates the omission of items from a code example or command format; the items are omitted because they are not important to the topic being discussed.
( )	In format descriptions, parentheses indicate that, if you choose more than one option, you must enclose the choices in parentheses.
[ ]	In format descriptions, brackets indicate optional elements. You can choose one, none, or all of the options. (Brackets are not optional, however, in the syntax of a directory name in a VMS file specification, or in the syntax of a substring specification in an assignment statement.)
{ }	In format descriptions, braces surround a required choice of options; you must choose one of the options listed.
boldface text	Boldface text represents the introduction of a new term or the name of an argument, an attribute, or a reason. Boldface text is also used to show user input in online versions of the manual.
italic text	Italic text emphasizes important information, indicates variables, and indicates complete titles of manuals. Italic text also represents information that can vary in system messages (for example, Internal error *number*), command lines (for example, /PRODUCER=*name*), and command parameters in text.
UPPERCASE TEXT	Uppercase text indicates a command, the name of a routine, the name of a file, or the abbreviation for a system privilege.
-	A hyphen in code examples indicates that additional arguments to the request are provided on the line that follows.
numbers	All numbers in text are assumed to be decimal, unless otherwise noted. Nondecimal radixes—binary, octal, or hexadecimal—are explicitly indicated.

BAD BLOCK LOCATOR Description

The Bad Block Locator Utility (BAD) locates bad blocks on a volume by testing whether data that is written onto blocks can be read back. When it finds a bad block, BAD writes the address of that block to the Software Detected Bad Block File (SDBBF).

To invoke BAD, enter the DCL command ANALYZE/MEDIA. To ensure that the device is not accessed by any other programs, you must first allocate the device with the DCL command ALLOCATE.

After you allocate the device, enter the DCL command MOUNT/FOREIGN. When the device is mounted foreign, the system does not recognize it as a Files–11 volume and BAD can execute. Table BAD–1 in Section 5 describes the devices that BAD can analyze.

To test the blocks on a volume, BAD does the following:

- Writes a test pattern to each block on the medium
- Reads the contents of the block into a buffer
- Compares the data read back with the data written

If the data does not compare exactly, a block cannot reliably store data.

When you run BAD to test a device (using the /EXERCISE qualifier), keep in mind the following:

- The device cannot be accessed by other programs.
- The device cannot be mounted as a Files–11 volume.
- The device is always purged by BAD's testing procedure; any information stored on the disk is destroyed.

Caution

There is no way to test the volume for bad blocks without destroying its contents. However, you can update the Detected Bad Block File (DBBF) without erasing the volume's contents by using the BAD qualifiers /NOEXERCISE and /BAD_BLOCKS.

1 Recording Bad Blocks

When BAD locates a bad block, it records the address of the block. Consecutive bad blocks are recorded as single entries for non-last-track devices. After it finishes testing the disk, BAD writes the addresses of the bad blocks into an area called the Detected Bad Block File (DBBF).

After BAD locates and records the bad blocks, enter the DCL command INITIALIZE to change the volume from unstructured format to Files–11 format and to allocate the faulty blocks to a special file on the volume called [000000]BADBLK.SYS. In this way, users are protected from accessing bad blocks for their files. For more information on Files–11 format and the INITIALIZE command, see the *OpenVMS DCL Dictionary*.

1.1 Location of the Detected Bad Block File

The location of the DBBF depends on whether the volume is a last-track device, that is, one that contains more than 4096 blocks (512 bytes per block). Last-track devices store bad block data on the last track of the disk.

The first half of the track is reserved for the Manufacturer's Detected Bad Block File (MDBBF). The MDBBF stores the bad blocks discovered by the manufacturer when the device was originally formatted.

The second half of the track is reserved for the Software Detected Bad Block File (SDBBF).

Non-last-track devices are devices that contain 4096 blocks (512 bytes per block) or less. These devices do not set aside the last track of the disk to store bad block information. Instead, BAD creates the DBBF on the last good block of the disk. There must be at least one reliable block in the last 256 blocks of the volume for BAD to generate the DBBF.

1.2 Format of the Detected Bad Block File

If the volume is a last-track device, each DBBF entry contains the cylinder, track, and sector address of the faulty block.

On volumes that are not last track, DBBF entries contain the number of bad blocks minus 1, and the logical block number (LBN) of the faulty block or sequence of faulty blocks. A single entry can address one bad block or several contiguous bad blocks.

The DBBF can contain a maximum of 126 entries. For both last-track and non-last-track devices, BAD terminates with an error message when the maximum number of entries is exceeded. However, the contents of the SDBBF on the medium remains intact.

2 Running BAD on Devices Converted to Non-Last-Track Format

If you run BAD in compatibility mode using the override (/OVR) qualifier on a device designated as last-track, the medium on the device is converted to non-last-track format. When you run BAD in native mode on such a device, BAD treats it as a last-track device and fails to find the Manufacturer's Detected Bad Sector File (MDBSF). Depending on which qualifiers you specify with the ANALYZE/MEDIA command, you can expect the following results when running BAD on a converted device:

- If you do not attempt to modify the medium on the device (for example, if you specify /NOEXERCISE /OUTPUT), BAD, upon failing to locate the MDBSF, attempts to find the Software Detected Bad Sector File (SDBSF) and produces an output listing using the non-last-track format.
- If you attempt to modify the medium preserving the SDBSF (for example, if you specify /EXERCISE=KEEP or /NOEXERCISE /BAD_BLOCKS), BAD terminates with the following fatal diagnostic message:

```
% BAD-F-MDBSFRFAIL, Failed to read the manufacturer's detected bad
sector file on DEVICENAME
```

Note that all attempts to preserve the SDBSF when modifying the medium on a converted device will fail. However, if you do not attempt to preserve the SDBSF (for example, by specifying /EXERCISE=KEEP), BAD recreates the MDBSF and SDBSF and continues testing. You could use the following command sequence to run BAD on a converted device, in this case DB1:

```
$ ANALYZE/MEDIA/OUTPUT=DB1TEST.ANL DB1:

ANALYZE/MEDIA                              19-APR-1990 08:16:53.97   PAGE 1
DB1: (DBA1:), SERIAL NUMBER: 0


SOFTWARE DETECTED BAD BLOCKS

LOGICAL BLOCK NUMBER    COUNT
   88308                  0

DEVICE DBA1:CONTAINS A TOTAL OF 340670 BLOCKS; 1 DEFECTIVE BLOCK DETECTED

ANALYZE/MEDIA/EXERCISE/OUTPUT=DB1TEST.ANL DB1:
```

3 Running BAD Interactively from Your Terminal

When you run BAD interactively from your terminal, use a sequence of commands similar to the following:

```
$ ALLOCATE DBA2:
DBA2: ALLOCATED
$ MOUNT/FOREIGN DBA2:
%MOUNT-I-MOUNTED     mounted on DBA2:
$ ANALYZE/MEDIA/EXERCISE DBA2:
$ INITIALIZE DBA2:
```

The ALLOCATE command requests the allocation of a specific disk drive, DBA2. The response from the ALLOCATE command indicates that the device was successfully allocated. The MOUNT/FOREIGN command mounts the disk volume as a foreign disk. The MOUNT command response indicates that DBA2 was successfully mounted. The ANALYZE/MEDIA command invokes BAD. Specifying DBA2 and the /EXERCISE qualifier causes BAD to analyze each block on this disk volume and to record the bad blocks. The INITIALIZE command reformats the volume as a Files–11 volume and allocates the bad blocks to a file on the volume called [000000]BADBLK.SYS. Once allocated, the bad blocks cannot be used by other files.

4 Running BAD from a Command Procedure

You can invoke BAD from a VMS command procedure. The following command procedure, NEWDISK.COM, invokes BAD and allocates, mounts, dismounts, and initializes the volume:

```
$! THIS COMMAND PROCEDURE EXERCISES AND INITIALIZES THE
$! SPECIFIED DISK.
$!
$! P1 IS THE DEVICE
$! P2 IS THE VOLUME LABEL
$!
$!
$ IF P1 .EQS. "" THEN $ INQUIRE P1 "DEVICE TO BE INITIALIZED"
$ IF P2 .EQS. "" THEN $ INQUIRE P2 "VOLUME LABEL"
$ ALLOCATE 'P1
$ MOUNT/FOREIGN 'P1
$ ANALYZE/MEDIA/EXERCISE 'P1
$ DISMOUNT/NOUNLOAD 'P1
$ INITIALIZE 'P1 'P2
$ EXIT
```

To run this command procedure, enter the following command:

```
$ @NEWDISK device volume-label
```

5 Table of Block-Addressable Devices

Table BAD–1 includes the following information for each device that BAD can analyze:

- Device model name
- Device code
- Device type
- Revolutions per minute
- Block-addressable surfaces per medium
- Cylinders per medium
- Bytes per track
- Total bytes per medium

Note

You do not need to run BAD on Mass Storage Control Protocol (MSCP) devices (for example, RX50 diskettes, and RA60, RA80, and RA81 disk drives). All MSCP devices appear to the operating system as a contiguous stream of logical blocks. If part of the disk goes bad, the drive itself revectors the bad block automatically.

Table BAD–1 Block-Addressable Devices

Model	Code	Type	RPM	Surfaces	Cylinders	Bytes/Track	Bytes/Medium
RB02	DQ	Cart	2400	2	512	10,240	10,485,760
RL02	DL	Cart	2400	2	512	10,240	10,485,760
RM03	DR	Pack	3600	5	823	16,384	67,420,160
RM05	DR	Pack	3600	19	823	16,384	256,196,608
RB80	DQ	Fix	3600	14	559	15,872	124,124,272
RM80	DR	Fix	3600	14	559	15,872	124,124,272
RP05	DB	Pack	3600	19	411	11,264	87,960,576
RP06	DB	Pack	3600	19	815	11,264	174,423,040
RP07	DR	Fix	3600	16[1]	630	25,600	516,096,000
RK06	DM	Cart	2400	3	411	11,264	13,888,512
RK07	DM	Cart	2400	3	815	11,264	27,540,480
RX01	DX	Flop	360	1	77	3,328	256,256
RX02	DX	Flop	360	1	77	3,328[2]	256,256
						6,656[3]	512,512
TU58[4]	DD	Cart	—	—	—	—	262,144

[1]The RP07 has 16 surfaces but 32 tracks per cylinder.

[2]The RX02 single-density diskette.

[3]The RX02 double-density diskette.

[4]The TU58 is a magnetic tape device that operationally resembles a disk device.

Device Types

Pack—disk pack
Cart—disk cartridge
Flop—flexible diskette
Fix—fixed disk

BAD BLOCK LOCATOR Usage Summary

The Bad Block Locator Utility (BAD) analyzes block-addressable devices and records the locations of blocks that cannot store data reliably.

Format

ANALYZE/MEDIA device

Parameter

device
Specifies the device containing the volume that BAD will analyze. The device name has the form ddcu: or logical-name.

Usage Summary

To invoke BAD, enter the command ANALYZE/MEDIA at the DCL prompt along with any parameters or qualifiers. Once invoked, BAD runs until completion. When BAD terminates, control is returned to the DCL command level.

To write the contents of the Detected Bad Block File (DBBF) to an output file, specify the /OUTPUT qualifier, as described in the following section.

BAD BLOCK LOCATOR Qualifiers

This section presents qualifiers for the ANALYZE/MEDIA command in alphabetical order. The qualifiers follow the standard rules of DCL syntax. Thus, you can abbreviate any qualifier or keyword as long as the abbreviation is not ambiguous. The asterisk and the percent sign can be used as wildcard characters unless otherwise noted.

/BAD_BLOCKS

Adds the specified bad blocks to the Detected Bad Block File (DBBF). If the /BAD_BLOCKS qualifier is specified along with the /EXERCISE qualifier, the medium is tested and the bad blocks are added to the DBBF.

Format

/BAD_BLOCKS [(=list)]

Keyword

list
Specifies codes for the bad block locations to be added to the DBBF.

If you do not specify a value for the /BAD_BLOCKS qualifier, BAD prompts as follows:

```
BAD_BLOCKS =
```

When it prompts, BAD reports any duplicate bad blocks. To terminate the prompting session, type CTRL/Z.

Note

The term *block* denotes a standard unit of 512 bytes, whereas the term "sector" denotes the physical size of the device sector, which is not always the same for all devices. For example, an RL02 has a sector size of 256 bytes, while an RK07 has a standard sector size of 512 bytes.

Valid bad block location codes follow. You can specify them in any integer combination or radix combination.

Code	Meaning
lbn	Specifies the logical block number (LBN) of a single bad block.
lbn:count	Specifies a range of contiguous bad blocks starting at the logical block number (LBN) and continuing for "count" blocks.
sec.trk.cyl	Specifies the physical disk address (sector, track, and cylinder) of a single bad sector. This code is valid only for last-track devices.
sec.trk.cyl:count	Specifies a range of bad sectors starting at the specified physical disk address (sector, track, and cylinder) and continuing for "count" sectors. This code is valid only for last-track devices.

Examples

1.
```
$ ANALYZE/MEDIA/BAD_BLOCKS=(4.4.4:3) DB1:
```

   The /BAD_BLOCKS qualifier in this example specifies a range of 3 bad sectors beginning at the physical disk address sector 4, track 4, cylinder 4. This range is added to the DBBF.

2.
```
$ ANALYZE/MEDIA/EXERCISE/BAD_BLOCKS=(2) DB1:
```

   The command in this example adds the bad block specification to the DBBF and then tests the medium. The bad block in this example is located at logical block number (LBN) 2.

3.
```
$ ANALYZE/MEDIA/EXERCISE/BAD_BLOCKS DB1:
BAD_BLOCKS = 2:3
BAD_BLOCKS = 4
% BAD-I-DUPBLKNUM, duplicate block number 4, already exists in SDBBF
-BAD-I-SRCLIN, the source input entry was 4
BAD_BLOCKS =
.
.
.
BAD_BLOCKS= CTRL/Z
```

   In this example, BAD prompts for bad block specifications. Note that, in prompt mode, BAD reports any duplicate bad blocks it detects.

/EXERCISE

Controls whether the medium should actually be tested. The default is /NOEXERCISE.

Format

/EXERCISE [=(keyword[,...])]

/NOEXERCISE

Keywords

FULL
Causes BAD to test the medium using three test patterns (0's, 1's, and "worst case") instead of the default single "worst case" pattern. The FULL keyword can be used only with /EXERCISE. Note that the "worst case" pattern always remains on media tested with the /EXERCISE qualifier.

KEEP
Ensures the preservation of the current SDBBF. The KEEP keyword is the default when /NOEXERCISE is specified.

NOKEEP
Causes BAD to create a new SDBBF. The NOKEEP keyword is the default when /EXERCISE is specified. This keyword cannot be used with the /NOEXERCISE qualifier.

PATTERN=(longword[,...])
Allows users to specify the value of a test pattern to be used as "worst case." Up to an octaword of test pattern data may be specified in decimal (%D), hexadecimal (%X), or octal (%O) radixes. The default radix is decimal.

The pattern is specified in longwords. If two or more longwords are specified, they must be enclosed in parentheses and separated by commas.

Examples

1. `$ ANALYZE/MEDIA/EXERCISE=FULL DB1:`

   The command in this example tests the medium using three test patterns. By default, a new SDBBF is created.

2. `$ ANALYZE/MEDIA/EXERCISE=KEEP DB1:`

   The command in this example tests the medium while preserving the current SDBBF.

3. `$ ANALYZE/MEDIA/EXERCISE=PATTERN=(%XFFEEFFEE,%XBADBADBA) DB1:`

   The command in this example specifies a hexadecimal test pattern two longwords in length.

4. `$ ANALYZE/MEDIA/NOEXERCISE/BAD_BLOCKS DB1:`

   The command in this example updates the DBBF without erasing the volume's contents.

/LOG

Specifies whether a message is sent to SYS$OUTPUT and to SYS$ERROR indicating the total number of bad blocks detected by BAD. The default is /NOLOG.

Format

/[NO]LOG

Example

```
$ ANALYZE/MEDIA/LOG DBB1:
Device DBB1: contains a total of 340670 blocks; 11 defective blocks detected.
```

The command in this example requests BAD to report the total number of bad blocks it detected on the device DBB1.

/OUTPUT

Specifies whether the contents of the DBBF are written to the specified file. If you omit the /OUTPUT qualifier, no output is generated.

When you specify /OUTPUT in conjunction with the /SHOW qualifier, the default keyword for the /SHOW qualifier is AFTER.

Format

/OUTPUT =[file-spec]

Keyword

file-spec
Identifies the output file for storing the results of the medium analysis. If you specify a file type and omit the file name, the default file name ANALYZE is used. The default file type is ANL. If you omit the file-spec, the results are output to SYS$OUTPUT.

In place of the file-spec, you may specify an output device. In this case, BAD writes the contents of the volume's DBBF to a file called ANALYZE.ANL and queues the file to the output device.

No wildcard characters are allowed in the file specification.

Examples

1. `$ ANALYZE/MEDIA/OUTPUT=BADDBBF.DAT DBA2:`

   The command in this example writes the contents of the DBBF from DBA2 to the output file BADDBBF.DAT. Note that because /NOEXERCISE is the default, the medium is not tested.

2. `$ ANALYZE/MEDIA/OUTPUT=LPB0: DBA2:`

   The command in this example writes the contents of the DBBF from DBA2 to the default file ANALYZE.ANL and queues the file to the print device LPB0.

/RETRY

Enables the device driver to retry soft errors. The /RETRY qualifier is used only in conjunction with the /EXERCISE qualifier. The default is /NORETRY.

Format

/EXERCISE /NORETRY

/EXERCISE /RETRY

Example

```
$ ANALYZE/MEDIA/EXERCISE/RETRY DBA0:
```

The command in this example directs the device driver to retry soft errors.

/SHOW

Lists the contents of the DBBF before and after the medium is exercised (tested).

Format

/SHOW [=(keyword[,...])]

Keywords

[NO]BEFORE,[NO]AFTER
Specifies whether the contents of the DBBF is listed before, after, or before and after the medium is exercised (tested). AFTER is the default.

Examples

1. `$ ANALYZE/MEDIA/EXERCISE/OUTPUT/SHOW=(BEFORE,AFTER) DBA3:`

   The command in this example lists the contents of the DBBF both before and after the disk DBA3 is exercised and directs the data to the current SYS$OUTPUT device.

2. `$ ANALYZE/MEDIA/EXERCISE/OUTPUT/SHOW DBA3:`

   The command in this example lists the contents of the DBBF only after the disk DBA3 is exercised and directs the data to SYS$OUTPUT.

Index