

**Val**

COLLABORATORS

	TITLE : Val		
ACTION	NAME	DATE	SIGNATURE
WRITTEN BY		October 27, 2024	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

# Contents

<b>1</b>	<b>Val</b>	<b>1</b>
1.1	Val V2.3 Documentation . . . . .	1
1.2	Copyright . . . . .	1
1.3	Introduction . . . . .	2
1.4	Starting Val . . . . .	3
1.5	Val's Cache . . . . .	7
1.6	Val Commands . . . . .	8
1.7	Problems . . . . .	13
1.8	Things To Do . . . . .	13
1.9	Program History . . . . .	14
1.10	Compiling Val . . . . .	14
1.11	Contacting the Author . . . . .	14

# Chapter 1

## Val

### 1.1 Val V2.3 Documentation

Val V2.3 Dec 1993 Copyright (C) 1988-93  
Andrew Kemmis - All Rights Reserved

#### TABLE OF CONTENTS

- Copyright
- Introduction
- Starting Val
- Val's Cache
- Val Commands
- Problems
- Things To Do
- Program History
- Compiling Val
- Contacting the Author

### 1.2 Copyright

Val V2.3 Dec 1993 Copyright (C) 1988-93  
Andrew Kemmis - All Rights Reserved

Val is supplied free of charge for personal use only. Commercial use or commercial distribution is prohibited without the direct consent of the author. Non-commercial distribution must include all files listed below and they must not be modified. Appropriate \*.info files may be added and/or modified to suit the distribution.

The files in this distribution are:

- Val - executable
- Val.guide - program documentation (this file)
- Val.c - main source code
- Val.q1 - qualifier parser input (Val.qi source)
- Val.qi - qualifier parser output (helps reading C source)

---

makefile - Aztec 'C' 3.6 makefile (for Val and others)

Val comes with NO WARRANTIES WHATSOEVER. The author is not responsible for any loss or damage arising from the use of this program; the user takes all such responsibility.

The source provided is also Copyright (C) 1988-93 Andrew Kemmis - All Rights Reserved. It is provided to the user purely and for no other purpose than to show how some of the program was written.

In a non-commercial distribution no charge may be made for Val, other than a nominal copy fee.

However, permission is expressly granted to Fred Fish to distribute Val on his cool collection of disks or his own CD collection disks.

Although Val is free to non-commercial users, any donations would be gladly accepted, preferably programs that you yourself have written and find useful. See `Contacting the Author` for more information.

One exception to this is: If you find that Val helps you under some dire circumstance where you would have been either unable to fix the problem without the help of the Val program, or you would have spent many more hours fixing the problem had it not been for Val, then consider sending me a small amount of money for the help Val has been for you. I will send you a more recent version of the program on floppy disk if the amount is \$25 US or more (or will send it when the more recent version is completed) with any other tools that I have fit to be distributed.

## 1.3 Introduction

Val is a disk partition validator. It SHOULD ONLY read your disk it does not attempt to fix it for you (i.e you can have it write-protected)

It should work with any 512 byte block structured virtual or physical disk partition i.e. it should work with any device that allows you to do any random `CMD_READ` of 512 bytes at multiples of 512 byte Offsets - this includes devices like PC MFM floppy disk devices as well. If you don't use the `SEEK` option then it will read the partition from the beginning to the end sequentially.

Val by-passes the FileSystem running on the device and does direct I/O to the device itself. However, the device must be structured to match either the Original File System (OFS) or the newer Fast File System (FFS).

It currently does not support Directory-Cache blocks (I don't use them and I wanted to try to get this out before Fish Disk 1000 - which at the time of writing this I don't know if I will make it or not.) It should properly support all other aspects of the OFS and the FFS. Of course if you find something that you know is correct and Val says it is wrong - then let me know and I'll fix it and release it ASAP. I'll even email you the fixed version if you have an Internet email address :-). See `Contacting the Author` for more information.

I actually first wrote this program when I got some disk error messages

---

from the Disk-Validator back before 1988 (I can't remember exactly when and I didn't add the headers to any of my source till Jan 1988.) Anyway, it came back with some cryptic message and a block number. I fiddled around with the disk with a program to edit it directly, from one of the Fish Disks back then, and managed to work out what was wrong with some file and just unlinked the file from the directory it was in. This of course took a number of days because I had to get the OFS documentation to work out what the disk was supposed to look like. Finally I had corrected the problem and mounted the partition again (the driver was not auto-mounting.) This time it validated away ... and came up with another error. Obviously someone at our favourite computer company decided that you should only ever get one error on a device or already have a tool to scan the device for problems if you did get an error. Neither of which were true. (OK there was DiskDoctor - but I think all that needs to be said about that is: it was purposefully left out of the 3.0 release of the operating system!) Thus I wrote Val.

Just a little aside note for the non-technical: if you have run Val on your disk and got no errors, then want to make sure the bitmap is valid do the following: Use a disk editor (like BlockEd) and modify the root block so the bitmap valid flag is zero (long word 78 - it should be 0xffffffff - don't forget to correct the checksum), then run DiskChange on the disk. Now the Disk-Validator will spin off for a while checking your disk and correcting the bitmap if it is wrong (but it will not tell you if it did change it.)

## 1.4 Starting Val

You can run Val on any disk (or partition) simply by typing:

```
Val /dev=disk:
```

However, disk: must be the Dos internal name for the partition e.g. DF0: may be valid but Workbench1.3: is not if that is DF0:'s name.

N.B. by default Val will inhibit the drive you are validating, so you may want to read the information about the INHIBIT parameter below before you run Val.

Val has a number of options available at runtime. The options do not work in the standard way as yet (it's just a library function - that can and should be changed in the future) You specify options by typing at least the unique letters for that option (case is ignored) (shown in uppercase further down) after a '/' and if it is a value option using an '=' followed by the value e.g. to run Val in seek mode displaying a count every 100 blocks read you could enter the following:

```
Val /Dev=hd0:/Seek/ShowReads/Count=100
Val /s/sh/c=100 /d=hd0:
Val /dDEVICE=HD0: /sEEK /Show /cO=100
```

This does bring about the following problem (feature) you cannot do:

```
Val /d=/d0: (if the device name starts with a '/')
```

---

The following is the output if you specify an invalid command line:

```
usAge: Val /Qualifiers
# Qual      Def      Min Max  Incl Excl Atts
a Quiet      false
b Device     Mandatory 2   77      1
c Poolsize   0x1000  0x0 0xffff
d Seek       false      h
e Maxdepth   100    10  1000 d    h
f SHowreads  false      h
g Count      1000    1   10000000 f    h
h Tracks     false      defg
i DOSnames   true
j Inhibit    true
k Fsshow     false      no
l CACHe      1    1   10000
m Statcache  false
n Ofs        false      ko
o FFs        false      kn
p FAilreq    true
q IGnoreerrors false
r Errlimit   -1   -1  2147483647
s Warnlimit  -1   -1  2147483647
t Retries    0    0   7777
u Linksallowed false
v DEBugcache false
w DEBUGLink  false
```

Def is the default value (a highlighted 'Mandatory' means no default - you must supply a value)

Min & Max are the minimum and maximum for numbers

Incl shows the qualifiers that must be specified if this one is

Excl shows the qualifiers that musn't be specified if this one is

Atts shows the special attributes ('l' means the value is converted to lowercase)

Numbers can be entered in Hex, (with a leading '0x'), Octal (with a leading '0') or Decimal

The options are as follows:

#### QUIET

if set true don't display startup message at runtime.  
/QUIET or /!QUIET

#### DEVICE

this qualifier is mandatory - you must specify it. It is just the device name of the partition you wish to validate i.e. it cannot be just an ASSIGN to the partition or just the name for the partition specified in the root block e.g. /DEVICE=hd0:

#### POOLSIZE

Val uses quite a lot of memory, but each time it needs some it is usually quite a small amount. This is handled by a library function that allocates memory in chunks and dishes it out bit by bit as required. This helps avoid memory fragmentation if other programs are running at the same time

as Val and allocating memory. PoolSize defines the size of the chunks it allocates e.g. /POOLSIZE=0x4000

#### SEEK

this determines the way Val validates the partition. If set false: read and analyse the entire disk into memory then scan the result in memory. If true: follow the disk structure starting at the ROOT BLOCK and seeking to each block in a depth-first search of the disk structure. /SEEK requires a limited amount of memory to check the partition. I have used Val regularly on a 98% full 500M partition using my old A1000 and it uses under 250K of memory to validate it. The cache uses 1K plus 512 bytes for each cache block of contiguous CHIP memory extra. /!SEEK requires approximately 1Meg of memory for each 15Meg of disk space. Each link block found takes up an extra 18 bytes of memory independent of the SEEK option. /SEEK will only read the blocks it thinks are in use. Thus if a partition is nearly empty /SEEK will often be much quicker to run.

#### MAXDEPTH

a figure somewhat related to the maximum depth of the directory structure of the partition and is allocated at runtime if running in SEEK mode. The default value of 100 should be big enough for most partitions. If the figure is too small you will get an error saying that you have exceeded the value; thus you should increase it e.g. /MAXDEPTH=200 (N.B. this requires that you also specify /SEEK)

#### SHOWREADS

if true every Count blocks read display a line showing how many have been read. /SHOWREADS or /!SHOWREADS

#### COUNT

the Count value used in SHOWREADS e.g /COUNT=100 (N.B. this requires that you also specify /SHOWREADS)

#### TRACKS

display a line showing how many blocks required have been read every time a track boundary is about to be crossed. /TRACKS or /!TRACKS (N.B. you cannot specify this with SEEK, MAXDEPTH, SHOWREADS or COUNT)

#### DOSNAMES

if true attempt to determine the full directory/file specification for a block if it gives an error or warning. /DOSNAMES or /!DOSNAMES

#### INHIBIT

if true specifies that the partition should be inhibited while using Val i.e. unavailable to any other process at the filesystem level (including Val and Shell I/O). /INHIBIT or /!INHIBIT

#### FSSHOW

if true display the Dos internal string containing the name of the File System. /FSSHOW or /!FSSHOW

#### CACHE

use the programs disk-cache code to speed up access. The default

---

value is 1 which means don't cache anything (except the current block - which will almost never get hit) I have found cache values in the range 75 to 150 to be most efficient on my devices but it will probably depend on the computer and device being used. e.g. /CACHE=100. See Val's Cache for more information.

#### STATCACHE

if true display the Cache statistics when Val has finished. /STATCACHE or /!STATCACHE. This will give you an idea of how appropriate the value you specified in /CACHE=x was. See Val's Cache for more information.

#### OFS

if true assume that the partition is OFS and don't try to determine what it is. /OFS or /!OFS

#### FFS

if true assume that the partition is FFS and don't try to determine what it is. /FFS or /!FFS

#### FAILREQ

if true, if a disk I/O read error occurs (after number of retries specified in /RETRIES) put up a requester allowing the user to abort the program cleanly every time RETRIES has exceeded.

#### IGNOREERRORS

this make most sense if you understand the code. Basically: if a block contains any internal errors turning this option on says to assume that pointers to other blocks are valid and continue checking the rest of the block (by default it is ignored) e.g. If a Directory block has a checksum error the program normally does not bother to check if the target files and directories are valid. /IGNOREERRORS or /!IGNOREERRORS

#### ERRLIMIT

if the specified number of errors are found, abort the validation there (you can pretend to be the stupid Disk-Validator by specifying /ERRLIMIT=1 so that it aborts at the first error it finds :-)

/ERRLIMIT=-1 means to never abort.

#### WARNLIMIT

if the specified number of warnings are found, abort the validation. /WARNLIMIT=-1 means to never abort.

#### RETRIES

This specifies the number of times to try to READ into the cache (or a single block if no cache specified,) if errors are returned, till the program should give up and return an error to the function that called for the requested the block (which will be marked I\_UNREADABLE) or to pop up a requester offering to abort the whole program if /FAILREQ is specified. N.B. an error reading into the cache will invalidate the entire cache and the cache code will attempt to read the next block requested as if the cache was empty.

#### LINKSALLOWED

if true, links do not give a warning message under OFS. /LINKS or /!LINKS

---

```

DEBUGCACHE
    if true, show some debug during cache code execution. /DEBUGCACHE
    or /!DEBUGCACHE

DEBUGLINK
    if true, dump internal link structures after finished validating
    partition. /DEBUGLINK or /!DEBUGLINK

```

## 1.5 Val's Cache

A sample output of the Cache Statistics (Except for these letters) is:  
 (command was: Val /Seek/Cache=150/Statcache/Dev=hd0:) v

```

Cache Statistics (Real Time) - Cache = 2:150          a
Blocks Requested.....      15934      b
Blocks Read.....            59521      c
Average Reads/Request..      3.74      d
Total 5secs Cache Code.      359510     e
Average 5sec/CodeReq...      23.94     f
Total 5secs Cache IOReq 121405856     g
Average 5sec/IOReq..... 132829.17     h
Total 5sec Cache Req... 121765366     i
Average 5sec/Request...   7641.86      j
Hits on Small Cache....      788      4.95%   k
Hits on Large Cache.... 14232 89.32%   l
Reads to Small Cache...      529      3.32%   m
Reads to Large Cache...      385      2.42%   n

```

The meaning of each line is:

- a) The specified cache - actually 2 blocks bigger than specified for storage of a small and a large cache depending on what is being read by Val
- b) Number of single block requests - all requests are for 1 block
- c) Actual number of blocks read (each request reads 2 or Cachesize blocks)
- d) c)/b) i.e. average number of blocks read per single block request
- e) microseconds spent in the cache hit part of the code
- f) average microseconds spent for each request that hit the cache  
N.B. this does not appear if e) is zero
- g) microseconds spent in the cache miss part of the code
- h) average microseconds spent for each request that missed the cache  
N.B. this does not appear if g) is zero
- i) e)+g) total microseconds spent in cache code
- j) this is the important number! j)/b) average time per block access  
i.e. divide by 1000 and you get your millisecond block access time  
the smaller this figure the better the cache size for your device  
this figure should very closely approximate to a quadratic function  
of the cache size N.B. this does not appear if e) and g) are zero
- k) number and % hits on small cache (first 2 blocks of cache)
- l) number and % hits on large cache (size as you specify)
- m) number and % reads on small cache
- n) number and % reads on large cache

The % figure in k), l), m) & n) is the % of all requests (i.e. all 4 add up to 100%)

As stated above, `j)` is the most important figure - it must be minimised - that is the whole point of the cache code. With `SEEK`, if the cache is too large the figure will increase as the size does (until you hit an order of magnitude of the full partition.) The point of the code is to do better than one block at a time reading, which it does do for any device that doesn't have a memory resident cache, so the point is to increase the size of the cache to make this figure bottom out. The code is not based on anything I have done at University or read anywhere. It is just a hack that worked well at the time and has stayed put.

Of course all these time figures depend on the way I have gotten the time (which isn't very accurate) but it works for the sake of trying to optimise the size of the cache. If however something else is running while `Val` is running OR an I/O request takes more than 1 second, the figures that come out will be neither meaningful nor valid.

If you don't use `SEEK` the cache will be hit at every block, thus a much larger cache can be successfully used - of course that limits the amount of available memory for the rest of the program, which can be at a premium when `SEEK` is switched off! (and don't forget that 1 second limit stated above for the `StatCache` output)

## 1.6 Val Commands

`Val` will check out the disk and output error messages and warnings according to any errors it may find on the disk with a reasonably concise message about what it thinks is wrong. It `SHOULD NOT WRITE` to the disk under any circumstances so don't be afraid to use it as often as you like. It just tells you what it thinks is wrong but will not even attempt to fix it. That's up to you (with a program like `BlockEd` or a disk recovery program) Thus you can regularly run `Val` on your hard disk overnight or when the computer is not busy and check the drive(s) for problems.

`Val` will find errors that the standard `Disk-Validator` will ignore; e.g. If a file header points to more or less blocks than the `filesize` represents then a warning will be displayed. This is especially useful if for some reason the `filesize` is wrong. The `Disk-Validator` will actually give the blocks past the `filesize` back to the free-block list if the `filesize` is too small; and you will probably lose any data beyond the block boundary after the `filesize`.

Writing `Val` actually made me want to use the `OFS` rather than the `FFS` for critical data for one very good reason. I know of no way to check that the data in an `FFS` file is actually corrupt or valid, unless the data itself contains some form of CRC (e.g. some archive programs) This can be seen to be very apparent in the following circumstance: I bought a soccer game a while back when the `LAMER EXTERMINATOR` virus was first going around here in Australia. The disk I bought actually had the virus on it because the shop was in the habit of playing the games before selling them and they had managed to get their machine infected with the virus. Unfortunately it managed to scribble on a few disks before I realised what was happening. It just wrote the word "LAMER" (with spaces) all over random? blocks. If these had been `FFS Data Blocks` they would actually have been valid data blocks! So any important data (e.g. source code) is still to this day on

---

OFS partitions on my hard disk (just in case :-)

A brief explanation of disk Error and Warning messages follows:

An example: "War: B 882 Fil Foo:x.y Block Count incorrect (par = 880)"

All Errors start with "Err: B" and Warnings start with "War: B"  
the following number specifies the block that gave the error and the next  
string specifies the type of block it is or should be:

```
Root      Root Block
BitMap    Bitmap Block
Dir, Fil, Lnk Directory, File Header, or any Link type block
Dir       Directory Block
Fil       File Header Block
Ext       File Extension Block
Dat       File Data Block
SoftLnk   Soft Link Block
DirLnk    Hard Directory Link Block
FilLnk    Hard File Link Block
```

If the DOSNAMES qualifier is true the next string will be the full  
Device:[Directory[/Filename]] related to the block.

Then follows the error message (below with parameter meanings and  
explanations) N.B. An incorrect message can come up due to another  
block required to validate this block also being wrong e.g. failing  
to read an OFS data block correctly can affect the calculated size  
versus the real size in a File Header block

The format below is:

1st line indented 1 space:

Error/Warning Message

2nd and subsequent indented 1 tab lines:

Meaning of %n arguments in message

Following non-indented lines:

Meaning of message

Block type wrong (par = %1)

where %1 is the parent block number

The type should be that specified at the beginning of the message (but it  
isn't)

Key wrong (par = %1)

where %1 is the parent block number

The key should be the value of the current block (for this block type)  
(except Root which should be 0)

Checksum wrong (par = %1)

where %1 is the parent block number

The block's checksum is incorrect

No Bitmap blocks present!

The Root block's first bitmap block is 0 (meaning no more bitmap blocks)

Parent pointer (%1) != parent block (%2)

where %1 is the actual pointer value

and %2 is the expected parent block number  
As it says!

Block Count incorrect (par = %1)  
where %1 is the parent block number  
The number of blocks pointed to by this block (on its own) doesn't match  
the count stored in the block (for File Header and File Extension blocks)

FirstData incorrect (should be = %1)  
where %1 is what the pointer should be  
The actual first data pointer doesn't match the value stored in the  
secondary copy of the first data pointer in the block (yes there are  
two of them)

Bytesize incorrect (calc = %1 actual = %2)  
where %1 is the filesize according to data blocks  
and %2 is the filesize in the file header  
The OFS bytesize doesn't match the filesize according to the data blocks  
If any of the data blocks show errors then this error may show up  
If the filesize has actually been adjusted incorrectly this will show an  
error (even though the Disk-Validator may not)

Header pointer (%1) != header block (%2)  
where %1 is the actual pointer value  
and %2 is the expected header lock number  
The OFS Data Block Key doesn't match the File Header block number

%1 pointer (%2) out of disk range!  
where %1 is the type of pointer (Next and Prev mean Link Pointers)  
and %2 is the actual pointer value  
The pointer specified is not in the range Reserved to the number of blocks  
on the partition - 1 (Reserved is an internal number usually 2)

Sequence number wrong (should be %1, is %2)  
where %1 is the expected sequence number  
and %2 is the actual sequence number  
On OFS data block has its sequence number wrong

Data block not full (only allowed in last data block!)  
An OFS data block says it is not full but is not the last block.

Bitmap Flag says Bitmap is invalid!  
Root block says this.

Hash table size != calculated size!  
The Root block specifies the size of the hash table in all subsequent  
blocks on the partition. Val only understands one particular block size  
so you may get this if your disk block size is not 512 bytes (but I don't  
know for sure) If your disk has standard 512 byte blocks, then the value in  
the Root Block is wrong.

Block type NOT unknown type!  
An FFS Data block should probably not match a normal block, but if it does  
by coincidence then ignore this message unless the block is actually owned  
by some other part of the disk (you should have another related error if  
this is the case - like the 'Already Used (FFS error)' error)

---

Next data block (%1) != Previous data block's (%2) Next data block (%3)  
where %1 is the next data block  
and %2 is the previous data block  
and %3 is the previous data block's next pointer  
The Previous OFS Data block (%2) said it's next block should be %3 but the  
File Header or File Extension said it was %1

Blocks name is invalid (BSTR size is 0 or > 30)  
The first byte of a BSTR specifies the size of a BSTR, for names these  
should be >0 and <31

Gave error (%1) on read (par = %2)  
where %1 is the Hex value return from the failed read  
and %2 is the parent block number  
The specified block failed to be read by Dos and returned the shown error

Total blocks incorrect (calc = %1 actual = %2)  
where %1 is the filesize according to data blocks  
and %2 is the filesize in the file header  
An FFS File has %1 blocks according to the File Header and File Extension  
blocks, however the filesize in the File Header means it should have %2  
blocks (%1 != %2) (again an error reading the data blocks can cause this  
error)

Pointed to by %1 Already used (FFS Error)  
where %1 is the block that points to this block  
When running Val on an FFS partition it builds up a bitmap of used blocks.  
If the same block is pointed to more than once, this error will come up on  
the block with %1 being the second and subsequent blocks that point to this  
already used block (and you will probably get other errors if it isn't an  
FFS Data block) This error does pose a major problem in that the owner of  
all blocks would have to be stored by Val to determine who else pointed to  
the block, not just a single bit to say it is used. For a 512Meg partition  
that would require 2Meg of memory just to store the table (which I don't  
have spare - maybe I should add this as an option soon :-). Anyway, to solve  
this problem I wrote a program called Find (which you probably should find  
with this distribution under another directory) See that program's  
documentation for how to use it to get more information on these errors.  
This error also means that either you are running Val in SEEK mode or the  
block pointed to is invalid, of an unknown type (FFS Data) or the Root block.

Already used (FFS Error)  
Same as above but the block matches a Root block type

Pointed to by %1 Already used by %2 (FFS Error)  
where %1 is the block that points to this block  
and %2 is the block found first to point to this block  
Similar to above, but since we are not in SEEK mode and the block type  
is not invalid, unknown type (FFS DATA) or Root block type we know which  
other block already points to this block. N.B. for an OFS data block you are  
told the File Header Block that points to the block, not the File Extension  
(assuming such a block is found on an FFS partition!)

Prev Link Pointer not 0  
All File Headers or Directory Blocks should have a Prev Link Pointer of 0  
Also Soft Links are not allowed to contain Hard Links (I presume)

---

Next Link Pointer not 0  
A Soft Link has a Next Pointer (I presume this is invalid)

Contains a link (but partition is OFS)  
OFS should not have links (but it does here!)

Prev Link Pointer should not be 0  
All Link blocks should point back to the (same) original Fil or Dir block

Prev (%1) Must be Fil or Dir (not %2)  
where %1 is this Links previous block pointer  
and %2 is the type of the previous block it currently points to  
The Previous block pointed to by this Link Block is not a File Header or Directory Block, it is a %2 type block (but it should be a Fil or Dir)

Next's (%1) Prev (%2) should be This  
where %1 is this Links next block pointer  
and %2 is the next Links previous block pointer  
This block does not have a previous link (thus should be the head of a Link list) BUT the next block it points to does not point back to this block (all Link blocks should point back to the head)

Next's (%1) Prev (%2) != This Prev (%3)  
where %1 is this Links next block pointer  
and %2 is the next Links previous block pointer  
and %3 is this Links previous block pointer  
This block is not a Link head, so it's prev pointer should be the same as it's next's prev pointer (they both should point to the Link head) BUT they have different prev pointers, which is incorrect

Prev Pointer's type (%1) not valid for this type  
where %1 is is the type of the Prev  
Given the type of this block, the Prev Link type should match: i.e. if this is a Hard Dir Link then the Prev should be a Dir. If this is a Hard Fil Link then the Prev should be a Fil (neither or which is true in your case)

Next Pointer's type (%1) not valid for this type  
where %1 is is the type of the Next  
Given the type of this block, the Next Link type should match: i.e. if this is a Hard Dir Link or a Dir then the Next should be a Hard Dir Link. If this is a Hard Fil Link or a Fil then the Next should be a Hard Fil Link (neither of which is true in your case)

Prev (%1) is not Linked  
where %1 is the block number of the Prev  
The block number %1 is either not a Hard Link, or it is a Dir/Fil with no Next Link pointer. N.B. if it was a Hard Link it would have given an error from above (Prev shouldn't be a Hard Link)

Next (%1) is not Linked  
where %1 is the block number of the Next  
The block number %1 is not a Hard Link, Dir or Fil. N.B. if it were a Dir/Fil it would give an error from above (Next should be a Hard Link)

---

## 1.7 Problems

I have not tried this program on many different device types. Below is the list which I have tried (and all have worked)

- Standard 880K low density drives
- A Meager Mega Ram Card with WD1002-05 and hard disks (on A1000)
- A590 SCSI hard-disk devices (on A1000)
- A1200 internal IDE device

I know you cannot use this program on the RAM: device or other same devices.

I have been using this program for a few years now so most parts of it should be pretty reliable (I suppose other than the latest additions: Links and a few of the last flags)

I do not know if this works with interleaved devices i.e. if I should be using the interleave value in calculating block offsets - you should get many errors immediately if this doesn't work, to avoid this first time try using /ERRLIMIT=10 and it should stop pretty quickly if everything is totally out of sync. However I don't think it is a problem?

I have recently found a problem with using the INHIBIT flag on one particular partition I have on my A590 SCSI drive that is using OFS. It is not the only partition of this type, but for some reason it is the only one that fails to cancel the INHIBIT on the partition (it pops up a requester asking to mount the partition and cannot cancel the INHIBIT.) Anyone who comes across this same problem, the simplest solution is to not use the INHIBIT flag on that partition. It is the only partition that exhibits this problem - but then again it could be due to some other bug in the program that affects my device structure and only this partition produces the desired problem. If anyone has come across such a problem before and fixed it - please let me know! See [Contacting the Author](#) for more information.

## 1.8 Things To Do

Any suggestions would be very welcome.

OK it really does need some nice icons (send me suggestions and if I like and use them I'll put your name in the docs)

Val does not yet handle Directory Cache Blocks under the extended FFS.

Val doesn't validate the contents of the bitmap (should be easy to implement however - but need to handle Directory Cache Blocks first so as not to get errors if they are present) If Val gives no errors then the Disk-Validator can correct your bitmap if it is wrong

It would be nice to check that all dates on the disk are valid and optionally complain if child dates are greater than parent dates (and parent != max child)

There is a lot of empty space in some blocks types that I do not check e.g. if should be zeroed (this may cause inconsistencies between OFS

---

and FFS however)

I have tried to keep the program 1.3 compatible (especially since my main hard-disk is actually connected to an A1000) so don't suggest anything that is above 1.3 that cannot be optional.

Also need to tidy up and release a whole collection of disk utilities I have written (in times of dire need and have used to solve the problems.)  
Look out for a few more, hopefully in the near future (like this one :-)

I have absolutely no idea what would be necessary to make this program work with disk blocks that were not 512 bytes in size (assuming that a full Root/Directory/FileHeader/etc. block is not just 512 bytes on this type of device spread over multiple blocks if <512 bytes or part of a block if >512)

An optional owner block list would be nice (see the 'Already used' messages)

## 1.9 Program History

OK there isn't much here - but maybe I'll be able to add to it as time goes on :-)

V2.3

First public release

(Finally added FFS Links and Boot Block File System sensing if you don't specify the File System and a few extra little flags)

V2.2 and below:

Pre-release

## 1.10 Compiling Val

Val has been written to compile under Aztec C V3.6a, however not all of my include files or library objects have been provided in the distribution thus you should not (will not be able to) compile the program yourself from the distribution.

## 1.11 Contacting the Author

I can be reached with comments, suggestions, bug reports etc.  
at the following address till around March 1994:

Andrew Kemmis Smith  
2/16 Nelson Street  
Penshurst NSW 2222  
Australia

or always via my parents place:

Andrew Kemmis Smith

---

32 Kenneth Street  
Longueville NSW 2066  
Australia

or by email:

and@praxa.com.au

---