FLASH! ProDOS 8 SUPPORTS FILE RECOVERY

Sandy delves into the ProDOS code that handles file deletion, destruction and truncation.

When man bites dog, that's news. When Apple Computer abandons its masochistic stance on file deletion, that's a miracle. Well, my dear readers, a latter-day saint moving among the programmers in Cupertino has restored order and sanity to ProDOS 8. Whereas PRODOS version 1.2 and its forebears irrevocably mangled files in the process of deleting them, PRODOS version 1.3 and its progeny preserve the integrity of deleted files. Now, programs to "undelete" files and to scavenge damaged disks may be written with relative ease, provided that you understand the anatomy of healthy and deleted files. To that end, this edition of D/L deals with file deletion and volume bit map manipulation by the ProDOS machine language interface (MLI), and the next installment of D/L introduces a utility that resurrects deleted ProDOS files.

Before commenting upon the disassembled ProDOS DELETE and DESTROY code, let's briefly review how information is stored on a disk. For more complete details, several references are available [1-3].

DISK ORGANIZATION

A block is the basic unit of data storage on ProDOS disks. Each block consists of two 256-byte pages. On magnetic media (floppy and hard disks), blocks 0-1 house the *loader code*, which transfers the PRODOS file from disk to random access memory (RAM). Most electronic disks (RAM disks) cannot be booted directly, and thus contain dummy loader blocks. Block 2 is the *key block* (first block) of the *volume directory*. Blocks 3-5 are also reserved for volume directory usage. The *volume bit map* (VBM) begins on block 6 in accord with a pointer in the volume directory header. Depending upon the size of the disk, the map extends a variable number of blocks. All other blocks are allocated and deallocated as files are created, expanded, truncated and deleted. The above block layout holds for all current versions of ProDOS.

The VBM keeps track of whether each block on the volume is free or reserved. Each VBM byte represents eight blocks. The high- and low-order bits correspond to the lowest and highest numbered blocks, respectively, as shown in **Example 1**.

A set bit denotes a free block, whereas a clear bit means that the block is reserved. **Example 1** shows the state of the second VBM byte: blocks 8-13 are occupied and blocks 14-15 are free.

EXAMPLE 1: Byte 1 of the Volume Bit Map

Bit	7	6	5	4	3	2	1	0
Block	8	9	A	B	C	D	E	F
State	0	0	0	0	0	0	1	1

Apple 5 1/4-inch disks are usually formatted for 280 blocks. Dividing this number by 8 gives a value of 35, the number of VBM bytes needed on a mini-floppy disk. Here, only a small portion of one VBM block is required. Apple 3 1/2-inch disks hold 1,600 blocks and demand 200 VBM bytes. Still, less than one VBM block is needed. A hard disk formatted for 10 megabytes contains 20,000 blocks and must have 2,500 bytes in a 5-block VBM. So, the larger the disk, the more space is occupied by the VBM.

FILE ORGANIZATION

Although many data types exist, files are grouped into two structural categories: directory and nondirectory files. A description of each variety follows.

Directory Files

The first four bytes of each directory file block contain the numbers of the preceding (bytes 0-1) and succeeding (bytes 2-3) blocks in the file. A zero value means that no backward or forward link exists.

FIGURE 1: File Entry

FIELD LENGTH			ENTRY OFFSET
1 byte	storage.type	name.length	$00
15 bytes	file.name		$01 : $0F
1 byte	file.type		$10
2 bytes	key.pointer		$11 $12
2 bytes	blocks.used		$13 $14
3 bytes	EOF		$15 : $17
4 bytes	creation		$18 : $1B
1 byte	version		$1C
1 byte	min.version		$1D
1 byte	access		$1E
2 bytes	aux.type		$1F $20
4 bytes	last.mod		$21 : $24
2 bytes	header.pointer		$25 $26

The key blocks of volume directory and subdirectory files house the *volume directory header* and *subdirectory header*, which detail many important attributes of the directories. A prior D/L installment contains graphic illustrations of headers [4]. Under current ProDOS convention, each header and file entry in a directory consists of 39 bytes, and 13 entries fit into one directory block. These values are obtained by reading the header and are not carved in stone.

Figure 1 illustrates the composition of a file entry. It is taken from the previously-noted article [4] and patterned after (interpretation: "stolen from") a chart in one of my favorite manuals. [1] The diagram will stand you in good stead as we dissect the file deletion code. Pay particular attention to the high-order nibble of the first byte in the file entry, which specifies one of five storage types:

```
D = Subdirectory file
3 = Tree file
2 = Sapling file
1 = Seedling file
0 = Deleted file
```

Since only four blocks are reserved for it, no more than 51 files can fit into a root directory. If you seem to be missing one file (i.e., 4 x 13 = 52), remember that the header counts as one entry. In contrast, the size of a subdirectory file is limited by available disk space and by a generous maximum of 65,535 files.

Nondirectory Files

Standard or nondirectory files hold various types of data and are organized quite differently from directory files. The three storage types detailed below are determined by the location of the end-of-file (EOF) marker, rather than by the amount of data in the files. A primer on sparse files explains this apparent discrepancy [5].

***Seedling File* (0 < = EOF < = $200)** This smallest file type does not exceed one block and cannot contain more than 512 ($200) bytes. The single *data block* is necessarily the key block of the seedling file.

***Sapling File* ($200 < EOF < = $20000):** When the EOF is moved beyond the 512th byte, the file has grown to sapling size. An *index block*, now the key block, is created to store the numbers of data blocks. Index blocks are segmented into two pages of 256 ($100) bytes apiece. The least significant byte (LSB) of a block number is saved in the first page, and the most significant byte (MSB) is held in the corresponding position of the second page. For example, if the first two data block numbers in an index block were $FF and $100, bytes 0-1 of the index block would contain FF and 00, respectively, and bytes 256-257 would hold 00 and 01, respectively. Get it? Be sure you've got it.

Because an index block can house the numbers of 256 ($100) data blocks, the maximal size of a sapling file is 131,072 ($20000) bytes.

***Tree Files* ($20000 < EOF < $1000000)** When sapling size is exceeded, a tree file is formed. A *master index block*, the new key block, records the numbers of up to 128 index blocks. Theoretically, the top size of a tree file is 16,777, 216 ($1,000,000) bytes, which translates to 16 megabytes. Since the final byte of a tree file is reserved for the EOF, maximum data size is really one byte less than just stated.

To create a file with 16 megabytes of potential space, type the following command from Applesoft:

BSAVE BIG.EMPTY.FILE,A$2000,L1,B$FFFFFF

Despite holding a single datum, the resulting sparse file is prepared to receive 16,777,215 data bytes, as shown under the ENDFILE column when the CATALOG command is issued.

Here is a *mini-quiz* for some of my more enthusiastic readers: Why must room be reserved for the EOF in a tree file but not in a sapling or seedling file? Drop me a note just to let me know that you're still alive and kicking.

BASIC INTERPRETER DELETE CODE

The DELETE command issued from BASIC causes the named file to be removed from the directory. Whereas many ProDOS BASIC interpreter (BI) commands are complex [6], DELETE is the simplest of all because it relies upon the machine language interface (MLI) to do all the dirty work (**lines 110-111**). If we are going to understand the mechanics of file deletion, we'll have to invade that bastion of ProDOS power, the MLI.

MLI STORAGE SPACE

The bulk of the ProDOS kernel resides in the first bank of the language card, also known as bank-switched memory. As seen in the "equate" section of **Listing 1**, the MLI is chock-full of storage areas. Several buffers pertinent to the DESTROY code are touched upon here:

1. *Zero Page I/O Storage* — When communicating with a disk device driver, critical data is stored in $42-$47. A while back, when we built a RAM disk in the pages of D/L, this process was outlined [7]. Because zero page data is used by the System Monitor and by many programs, the MLI saves these locations when called and restores them on exit.
2. *File Control Block (FCB) Table* — In keeping with the maximum number of open files, up to eight 32-byte active FCBs may exist in the 256-byte FCB Table. Each FCB stores data relating to file identification, composition, and location.
3. *Volume Control Block (VCB) Table* — Again, a 256-byte area is segmented into eight 32-byte VCBs. Volume names and other data are held in each VCB.
4. *Volume Bit Map Block Buffer* — A 512-byte buffer is reserved for an image of the current VBM block.
5. *Primary Block Buffer* — This 2-page multipurpose buffer usually contains an image of a file block. For directory files, it is used to manipulate header and file entry information. For nondirectory files, it holds an image of an index block.
6. *Variable Data Area* — File header and ID data are saved here.

7. *File Entry Buffer* — This 39-byte segment houses an image if it's a file entry.
8. *Variable Work Area* — Much of the data in this work area is detailed in **lines 78-103.**

MLI DESTROY CODE

Our tale of destruction begins at **line 400** of **Listing 1**, the entry point to the MLI DESTROY command. After securing the file entry and copying it to the file entry buffer, data is extracted from the file entry and errors are reported (**lines 400-401**). If an unused FCB can be grabbed (**lines 402-404**), the work area is told that no free blocks are needed (**lines 405-407**), and the count of free blocks on the current volume is checked (**lines 408-409**). Since no blocks are requested, the code in **lines 410-411** should be bypassed. If the destroy bit of the access code [4] is not enabled, an error is flagged (**lines 412-416**). If the file is not locked, the integrity of the disk device is ensured (**lines 417-419**). After transferring data from the file entry buffer to the work area (**lines 420-423**), the *storage.type* nibble is tested. Directory file entries are routed to **line 477**, while standard file entries are passed to **line 435**.

A latter-day saint moving among the programmers in Cupertino has restored order and sanity to ProDOS 8.

DESTROYING DIRECTORY FILES

After rechecking for a directory file storage.type (**lines 477-478**), **line 479** calls RDVBMBLK (**lines 278-296**) to read the appropriate VBM block into the VBM block buffer. This subroutine first indexes the current VCB and uses CKPTVBM (**lines 306-313**) to ensure that the contents of the VBM block buffer are written to disk if another VBM block is needed. Should a new VBM block be required, DORDVBM (**lines 317-347**) resets values in the VCB and sets the zero page I/O storage locations for a direct READ call to the disk device driver via DOIO (**lines 377-394**). At this point, a VBM block and the key block of the directory file are in their respective block buffers.

Because unlocked directory files can be deleted only if they contain no active file entries, an empty target file is ensured (**lines 487-492**). If the file is deletable, the first byte (i.e., *storage.type/name.length*) of the directory header is zeroed (**line 493**), and the key directory block is written back to disk (**line 494**). If the directory encompasses more than one block (**lines 496-498**), the key block is freed in the VBM (**line 502**) and the block number designated by the forward link is read into the primary block buffer. Iteration of this process occurs until all blocks held by the directory file are marked free in the VBM. FRVBMBLK, the subroutine that releases blocks, is considered in a separate section. If the directory file occupies a single block (**lines 499-500**), control passes to **line 447**. There the target VBM bit is turned on, thus releasing the block, and the first byte of the file entry is zeroed. We'll discuss this further in the following paragraphs.

Destroying Nondirectory Files

After saving the *storage.type* (multiplied by 16) of the file (**line 435**), several locations in the work area are zeroed (**lines 436-440**), the byte offset into the block is maximized (**lines 441-442**), and the Destroy flag is enabled (**line 443**). **Line 444** calls TRUNCEOF (**lines 528-538**), the subroutine that shortens or destroys files, depending upon the state of the Destroy flag. TRUNCEOF determines the storage type of the target file and routes flow to the correct handler. In this article, we shall concentrate on destruction, not truncation, although the full code and commentary are provided in **Listing 1**. Keep in mind that, when a standard file is destroyed, the first byte in the file entry is zeroed, index blocks are altered, and the file count is decremented in the header of the parent directory — but not a single datum of file content is obliterated. If this were not so, file recovery would be impossible.

If a seedling file is found, TRNCSEED (**lines 643-662**) reads the key block, a data block, into the primary block buffer. Because the byte offset into the block points beyond the last block byte, the contents of the data block remain intact. This would not be the case if truncation were taking place.

If the target file is a sapling, TRNCSAP (**lines 616-639**) reads the key block, an index block, into memory and calls FREIXBL1 (**lines 699-723**) to do the hatchet job. Rather than zero the entries in the index block as did version 1.2 of the MLI, the latter subroutine swaps the MSBs and LSBs, so that the MSBs appear in the first page of the index block and the LSBs occupy the second page.

Once again, data is not destroyed, so deleted files can be reconstructed. FREIXBL1 begins by saving the entry block number on the stack and freeing the indexed data blocks in the VBM. DOIXBYT (**lines 727-736**) swaps the MSB and LSB bytes in the index block.

In version 1.3 of PRODOS, a 65C02/65802/65816 instruction snuck into the listing (**line 730**), causing the operating system to bomb on 6502 machines. Version 1.4 squashes this bug simply by substituting a 6502 opcode.

FREIXBL1 continues swapping index block bytes until all but the first entry are processed. After writing the index block back to disk, the initial entry is swapped by DEMFITYP (**lines 672-694**), a subroutine that demotes the file type from tree to sapling, sapling to seedling, or seedling to deleted.

When destroying a tree file, TRNCTREE (**lines 544-612**) is invoked. Because sparse files contain discontinuous data, all 128 potential index blocks are examined in the master index block. The latter block is read into memory and the numbers of the active (nonzero) subindex blocks are stored in the work area device table until the table is full (holds eight block numbers) or the EOF is reached. Each subindex block is:

1. Read into memory
2. Altered by FREIXBLK (**lines 698-723**), which swaps all entries in the subindex block
3. Written back to disk

When all entries in the work area device table have been handled, the process is repeated until each and every subindex block has been inverted. TRNCTREE ends by swapping all entries in the master index block and exits via TRNCSAP1 and TRNCSEE1.

When a standard file is destroyed, not a single datum of file context is obliterated.

VOLUME BIT MAP MANIPULATION

Because of its importance for authors of ProDOS utilities (especially me), I have included code that allocates blocks in the VBM as well as the subroutine used by the DESTROY Command Handler to free blocks in the VBM.

Freeing a Block in VBM

FRVBMBLK gets the ball rolling by storing the MSB and LSB of the block to be freed in VBMSRCH and the stack, respectively (**lines 119-120**). After ensuring that the number of the target block does not exceed the number of blocks on the disk (**lines 121-123, 125**), the bit position of the target block within the target byte is determined. This is done by using the three low-order bits of the target LSB as an index into a table of byte masks (**line 742**). The result is saved in VBMBIT (**lines 124, 126-130**).

With the target MSB and LSB now in VBMSRCH and the Accumulator, respectively, the block number is divided by eight to find the target byte position in the VBM (**lines 131-137**). Representing the byte offset in the target VBM page, the resulting LSB is stored in VBMBYOFS (**line 138**). The MSB divided by two denotes the block offset in the VBM, which remains in VBMSRCH (**line 139**). The page of the target VBM block is determined by picking up the Carry bit from the divide operation and saving it in VBMBUFPG (**line 140**).

With the block, page, byte and bit offsets of the target VBM block tucked away in the work area, the rest is not difficult. If the last-used VBM block holds our target bit (**lines 144-148**), control passes to **line 160**; otherwise, the last-used block is checkpointed and the target VBM block is read into memory (**lines 149-156**). Using VBMBUFPG, VBMBYOFS and VBMBIT, the target map bit is turned on (**lines 160-168**) and the VBM flag is set high to indicate that, when the next checkpoint is performed, this block must be written to disk (**lines 169-171**). After incrementing the count of freed (not free) blocks in the work area (**lines 172-174**), the Carry flag is cleared to signal successful liberation of a block in the VBM (**line 175**), and FRVBMBLK returns to its caller.

Now, programs to "undelete" files and to scavenge damaged disks may be written with relative ease.

Reserving a Block in VBM

ALVBMBLK reads the block pointed to by the VCB into the VBM block buffer (**lines 185-186**) and searches for the first byte that contains a set bit indicating an unused block (**lines 189-198**). This is the target VBM byte. If the current VBM block is full (**lines 199-201**), GETVBMBL (**lines 262-274**) checkpoints the block and reads the next VBM block into memory. If the end of the VBM has been reached, a VOLUME FULL error code is returned.

Calculating the block number represented by the first set bit in the target byte is the inverse of the process detailed in the prior section. Using two bytes of a work area Accumulator, multiplying the byte offset plus page offset by eight gives the base number of the block represented by the target byte (**lines 206-216**). To find the exact block number, a set Carry flag is used as a byte marker (**line 220**), and bits in the target byte are rotated left until a set bit pops into the Carry flag (**lines 221-229**). With each rotation, the base block number is bumped by one. When the free block is detected, a series of right shifts restores the target byte to its original form except that the target bit is now clear, signaling a reserved block (lines 230-237).

ALVBMBLK ends by setting the VBM flag (**lines 235-240**), subtracting one from the free (not freed) block count in the VCB (**lines 244-251**), and returning with the allocated block number in the work area Accumulator (**lines 255-258**).

AULD LANG SYNE

We meet again in 1988, the year of the giant RAM. At that time I will present a ProDOS file recovery utility that you will want to keep ever at hand. I wish you, me, and the staff at *Nibble* a happy and fulfilling New Year.

REFERENCES

1. *ProDOS 8 Technical Reference Manual*, Reading, MA: Addison-Wesley Publishing Company, Inc. 1987.
2. Little, Gary. *Apple ProDOS: Advanced Features For Programmers*. Bowie, MD: Brady Communications Company, Inc. (a Prentice-Hall Publishing Company). 1985.
3. Worth, Don, and Pieter Lechner, *Beneath Apple ProDOS*. Chatsworth, CA: Quality Software. 1984.
4. Mossberg, S. "Disassembly Lines: The CAT and CATALOG Commands." *Nibble*, May 1986, pp. 114-128.
5. Mossberg, S. "Apple Tutorial: Sparse Files." *Nibble*, June 1987, pp. 72-87.
6. Mossberg, S., *Dissassembly Lines: Vol. 4*. Concord, MA: Nibble Publications. 1986.

LISTING 1: DESTROY

Note: This code already exisits in the ProDOS MLI. There is no need to type it in.

```
1     ****************************************
2     *              DESTROY                 *
3     *         DELETING ProDOS FILES        *
4     *          MLI version 1.3/1.4         *
5     *            BI version 1.1            *
6     *      Interpreted by Sandy Mossberg   *
7     *        * Merlin Pro                  *
8     *          Copyright (C) 1987          *
9     *          by MicroSPARC, Inc.         *
10    *          Concord, MA  01742          *
11    *                                      *
12    ****************************************
13             XC                   ;handle 65C02 opcodes
14
15    * Zero Page (ZP) I/O Locations for Disk Device Driver:
16
17    DOCMDNUM =     $42            ;command code
18    DOSLTDRV =     $43            ;device code (DSSS 0000)
19    DOBUFPTR =     $44            ;buffer pointer
20    DOBLKNUM =     $46            ;block number
21
22    * BASIC Interpreter (BI) Global Page:
23
24    GOSYSTEM =     $BE70          ;execute call to MLI
25
26    * System Global Page
27
28    SYSERR   =     $BF09          ;system error handler
29    SYSDEATH =     $BF0C          ;system death handler
30    SERR     =     $BF0F          ;error code
31    DEVNUM   =     $BF30          ;device code (DSSS 0000)
32
33    * File Control Block (8 FCBs in FCB Table):
34
35    WRITFLG  =     $D81C          ;write flag (MI=write.PL=no wrt)
36
37    * Volume Control Block (8 VCBs in VCB Table):
38
39    VCUNUM   =     $D910          ;unit number of volume
40    VCTOTBLK =     $D912          ;total block count in volume
41    VCFREBLK =     $D914          ;free block count in volume
42    VCVBMOFS =     $D91A          ;block offset to multiblock VBM
43    VCNXTVBM =     $D91C          ;next VBM block to get
44
45    * Volume Bit Map (VBM) Block Buffer:
46
47    VBMBUF   =     $DA00          ;VBM Buffer
48
49    * Primary Block Buffer:
50
51    PBLKBUF  =     $DC00          ;primary block buffer
52    PBFWDPTR =     $DC02          ;forward pointer
53    PBSTYPNL =     $DC04          ;storage.type/name.length
54    PBFILCNT =     $DC25          ;file.count
55
56    * MLI Proper:
57
58    TOBLKIO  =     $DEE4          ;perform I/O via device handler
59    UPDATDIR =     $E4B2          ;update directory
60    GFILENT  =     $E593          ;get file entry
61    GFREBLK  =     $E959          ;get free block(s) if available
62    GETFCB   =     $EF9B          ;get a FCB
63    STATCALL =     $F43E          ;make status call
64
65    * Variable Data Area:
66
67    VDFILCNT =     $FE47          ;file count
68
69    * File Entry Buffer:
70
71    FESTYPNL =     $FE53          ;storage.type/name.length
72    FEKEYPTR =     $FE64          ;key.pointer
```

```
                ACCESS   =      $FE71        ;access

                * Variable Work Area:

                ACC      =      $FE7A        ;4-byte accumulator
                VCBOFFS  =      $FE85        ;offset into VCB table
                FCBOFFS  =      $FE86        ;offset into FCB table
                FBLKNEED =      $FE88        ;number of free blocks required
                FCBOPFLG =      $FE8B        ;FCB open flag (0=open,1=closed)
                VBMBIT   =      $FE8F        ;bit to free in target VBM byte
                VBMSRCH  =      $FE90        ;number of VBM blocks to search
                YSAV     =      $FE91        ;save Y-Reg
                VBMBYOFS =      $FE96        ;VBM byte offset in page
                VBMPGOFS =      $FE97        ;VBM page offset
                VBMBUFPG =      $FE98        ;VBM buffer page
                VBMFLG   =      $FE99        ;VMB flag (MI=write,PL=no write)
                VBMDVNUM =      $FE9A        ;VBM device code
                VBMBLKNM =      $FE9B        ;VBM block number
                VBMBLOFS =      $FE9D        ;block offset for multiblock VBM
                IOTFRFLG =      $FEA6        ;I/O transfer-occurred flag
                KYBLKPTR =      $FEB3        ;new key block pointer
                STORTYP  =      $FEB5        ;new storage type
                VWFREBLK =      $FEB6        ;count of freed blocks
                EOFBLKNM =      $FEB8        ;EOF block number (MSB/LSB)
                EOFBLKOF =      $FEBA        ;EOF byte offset into block
                                             ;MSB=page,LSB=offset into page
                EOFMIX   =      $FEBC        ;EOF master index counter
                DEVTBLIX =      $FEBD        ;index into Device Table (below)
                DEVTBL   =      $FEBE        ;holds #s of 8 blocks to free
                DSTRYFLG =      $FEEB        ;destroy flag (1=set,0=clear)

                *=================================================
                * BI DELETE COMMAND HANDLER:         >>>  BI CODE (v1.1)
                *=================================================
                         ORG    $AD7D        ;in low RAM

AD7D: A9 C1     DELETCMD LDA    #$C1         ;DESTROY code
AD7F: 4C 70 BE           JMP    GOSYSTEM     ;execute command
                *=================================================
                * FREE A BLOCK IN VOLUME BITMAP:    >>> MLI CODE (v1.3/1.4)
                *=================================================
                         ORG    $EA1A        ;in 1st bank of "language card"

                * Calculate target byte and bit in VBM:

EA1A: 8E 90 FE  FRVBMBLK STX    VBMSRCH      ;save target block number (MSB)
EA1D: 48                 PHA                 ;save target block number (LSB)
EA1E: AE 85 FE           LDX    VCBOFFS      ;index current VCB
EA21: BD 13 D9           LDA    VCTOTBLK+1,X ;compare total blocks on
EA24: CD 90 FE           CMP    VBMSRCH      ; volume with number of
EA27: 68                 PLA                 ; blocks requested
EA28: 90 6E              BCC    FRVBM6       ;block # larger than volume size
EA2A: AA                 TAX                 ;save target block number (LSB)
EA2B: 29 07              AND    #7           ;calculate bit number within
EA2D: A8                 TAY                 ; target block VBM byte
EA2E: B9 F4 FD           LDA    VBMSKTBL,Y   ; using bit lookup table
EA31: 8D 8F FE           STA    VBMBIT       ;save bit position in VBM block
EA34: 8A                 TXA                 ;with target block number in
EA35: 4E 90 FE           LSR    VBMSRCH      ; VBMSRCH (MSB) and A-reg (LSB)
EA38: 6A                 ROR                 ; divide by 8 to find target
EA39: 4E 90 FE           LSR    VBMSRCH      ; byte position in VBM
EA3C: 6A                 ROR
EA3D: 4E 90 FE           LSR    VBMSRCH      ;we now have byte position (MSB
EA40: 6A                 ROR                 ; and LSB)
EA41: 8D 96 FE           STA    VBMBYOFS     ;save byte offset in our block
EA44: 4E 90 FE           LSR    VBMSRCH      ;convert to block offset in VBM
EA47: 2E 98 FE           ROL    VBMBUFPG     ;save page in target block

                * Get target VBM block into buffer:

EA4A: 20 43 EB           JSR    RDVBMBLK     ;read VBM block
EA4D: B0 48              BCS    FRVBM5       ;read error
EA4F: AD 9D FE           LDA    VBMBLOFS     ;get block offset
EA52: CD 90 FE           CMP    VBMSRCH
EA55: F0 16              BEQ    FRVBM1       ;at required block
EA57: 20 76 EB           JSR    CKPTVBM      ;not proper block so checkpoint
EA5A: B0 3B              BCS    FRVBM5       ;error
EA5C: AD 90 FE           LDA    VBMSRCH
EA5F: AE 85 FE           LDX    VCBOFFS      ;index current VCB and store
EA62: 9D 1C D9           STA    VCNXTVBM,X   ; VBM block offset in VCB
EA65: AD 9A FE           LDA    VBMDVNUM     ;get device code for VBM
EA68: 20 87 EB           JSR    DORDVBM      ;read VBM block
EA6B: B0 2A              BCS    FRVBM5       ;read error

                * Free a block in VBM buffer:

EA6D: AC 96 FE  FRVBM1   LDY    VBMBYOFS     ;get byte offset into block
EA70: 4E 98 FE           LSR    VBMBUFPG     ;CC=1st page,CS=2nd page
EA73: AD 8F FE           LDA    VBMBIT       ;get bit position in byte
EA76: 90 08              BCC    FRVBM2       ;we're in 1st page
EA78: 19 00 DB           ORA    VBMBUF+256,Y ;2nd page: free target block
EA7B: 99 00 DB           STA    VBMBUF+256,Y ; by setting target bit in VBM
EA7E: B0 06              BCS    FRVBM3       ;always
EA80: 19 00 DA  FRVBM2   ORA    VBMBUF,Y     ;1st page: free target block
EA83: 99 00 DA           STA    VBMBUF,Y     ; by setting target bit in VBM
EA86: A9 80     FRVBM3   LDA    #$80         ;indicate that on checkpoint
EA88: 0D 99 FE           ORA    VBMFLG       ; this block should be
EA8B: 8D 99 FE           STA    VBMFLG       ; written to disk
EA8E: EE B6 FE           INC    VWFREBLK     ;add to freed block count
EA91: D0 03              BNE    FRVBM4
EA93: EE B7 FE           INC    VWFREBLK+1
EA96: 18        FRVBM4   CLC                 ;signal no error
EA97: 60        FRVBM5   RTS
EA98: A9 5A     FRVBM6   LDA    #$5A         ;VBM error code
EA9A: 38                 SEC                 ;signal error
EA9B: 60                 RTS
                *-----------------------------
                * ALLOCATE BLOCK IN VOLUME BITMAP:
                *-----------------------------
                * Get first set bit (i.e. free block) in VBM:

EA9C: 20 43 EB  ALVBMBLK JSR    RDVBMBLK     ;read VBM block
EA9F: B0 23              BCS    :4           ;read error
EAA1: A0 00     :1       LDY    #0           ;index 1st byte
```

```
EAA3: 8C 98 FE           STY    VBMBUFPG     ;set 1st page
EAA6: B9 00 DA  :2       LDA    VBMBUF,Y     ;get VBM byte in 1st page
EAA9: D0 1A              BNE    :5           ;free block found in this byte
EAAB: C8                 INY
EAAC: D0 F8              BNE    :2           ;loop back until 1st page done
EAAE: EE 98 FE           INC    VBMBUFPG     ;bump to 2nd page
EAB1: EE 97 FE           INC    VBMPGOFS     ;bump page offset into VBM
EAB4: B9 00 DB  :3       LDA    VBMBUF+256,Y ;get VBM byte in 2nd page
EAB7: D0 0C              BNE    :5           ;free block found in this byte
EAB9: C8                 INY
EABA: D0 F8              BNE    :3           ;loop back until 2nd page done
EABC: EE 97 FE           INC    VBMPGOFS     ;bump page offset into VBM
EABF: 20 28 EB           JSR    GETVBMBL     ;get another VBM block
EAC2: 90 DD              BCC    :1           ;no error so loop back
EAC4: 60        :4       RTS                 ;error

                * Calculate block number represented by 1st set VBM bit:

EAC5: 8C 96 FE  :5       STY    VBMBYOFS     ;save byte offset into VBM page
EAC8: AD 97 FE           LDA    VBMPGOFS     ;(page.offset+byte.offset)*8+
EACB: 8D 7B FE           STA    ACC+1        ; bit position=block number
EACE: 98                 TYA                 ; represented by target VBM bit
EACF: 0A                 ASL
EAD0: 2E 7B FE           ROL    ACC+1
EAD3: 0A                 ASL
EAD4: 2E 7B FE           ROL    ACC+1
EAD7: 0A                 ASL
EAD8: 2E 7B FE           ROL    ACC+1        ;ACC+1=block number (MSB)
EADB: AA                 TAX                 ;X=incomplete block number (LSB)

                * Allocate block by clearing 1st set bit in VBM:

EADC: 38                 SEC                 ;mark byte position to return to
EADD: AD 98 FE           LDA    VBMBUFPG     ;get buffer page
EAE0: F0 05              BEQ    :6           ;on 1st buffer page
EAE2: B9 00 DB           LDA    VBMBUF+256,Y ;get target byte from 2nd page
EAE5: B0 03              BCS    :7           ;always
EAE7: B9 00 DA  :6       LDA    VBMBUF,Y     ;get target byte from 1st page
EAEA: 2A        :7       ROL                 ;rotate the sucker
EAEB: B0 03              BCS    :8           ; until a set bit pops out
EAED: E8                 INX                 ;bump block number (LSB) with
EAEE: D0 FA              BNE    :7           ; each shift & always loop back
EAF0: 4A        :8       LSR                 ;shift back to original position
EAF1: 90 FD              BCC    :8           ; thus clearing set target bit
EAF3: 8E 7A FE           STX    ACC          ;ACC=block number (LSB)
EAF6: AE 98 FE           LDX    VBMBUFPG     ;get buffer page
EAF9: D0 05              BNE    :9           ;on 2nd buffer page
EAFB: 99 00 DA           STA    VBMBUF,Y     ;return altered byte to 1st page
EAFE: F0 03              BEQ    :10          ;always
EB00: 99 00 DB  :9       STA    VBMBUF+256,Y ;put altered byte on 2nd page
EB03: A9 80     :10      LDA    #$80         ;indicate that on checkpoint
EB05: 0D 99 FE           ORA    VBMFLG       ; this block should be
EB08: 8D 99 FE           STA    VBMFLG       ; written to disk

                * Do VCB housekeeping:

EB0B: AC 85 FE           LDY    VCBOFFS      ;index current VCB
EB0E: B9 14 D9           LDA    VCFREBLK,Y   ;subtract one from free
EB11: E9 01              SBC    #1           ; block count in VCB
EB13: 99 14 D9           STA    VCFREBLK,Y
EB16: B0 08              BCS    :11
EB18: B9 15 D9           LDA    VCFREBLK+1,Y
EB1B: E9 00              SBC    #0           ;pick up carry
EB1D: 99 15 D9           STA    VCFREBLK+1,Y

                * Return with allocated block number in accumulator:

EB20: 18        :11      CLC                 ;signal no error
EB21: AD 7A FE           LDA    ACC          ;block number (LSB)
EB24: AC 7B FE           LDY    ACC+1        ;block number (MSB)
EB27: 60                 RTS
                *-----------------------------
                * GET NEXT VOLUME BITMAP BLOCK:
                *-----------------------------
EB28: AC 85 FE  GETVBMBL LDY    VCBOFFS      ;index current VCB
EB2B: B9 13 D9           LDA    VCTOTBLK+1,Y
EB2E: 4A                 LSR                 ;calculate total number
EB2F: 4A                 LSR                 ; of blocks in VBM
EB30: 4A                 LSR
EB31: 4A                 LSR
EB32: D9 1C D9           CMP    VCNXTVBM,Y   ;have we exceeded this number?
EB35: F0 3B              BEQ    DFULLERR     ;yes, so disk full error
EB37: B9 1C D9           LDA    VCNXTVBM,Y   ;no,
EB3A: 18                 CLC                 ; so
EB3B: 69 01              ADC    #1           ; add one to indicate
EB3D: 99 1C D9           STA    VCNXTVBM,Y   ; next VBM block to get
EB40: 20 76 EB           JSR    CKPTVBM      ;checkpoint old block
                *-----------------------------
                * READ VOLUME BITMAP BLOCK:
                *-----------------------------
EB43: AC 85 FE  RDVBMBLK LDY    VCBOFFS      ;index current VCB
EB46: B9 10 D9           LDA    VCUNUM,Y
EB49: CD 9A FE           CMP    VBMDVNUM
EB4C: F0 0E              BEQ    :1           ;VBM for this unit already read
EB4E: 20 76 EB           JSR    CKPTVBM      ;checkpoint VBM of another unit
EB51: B0 1E              BCS    RTS10        ;error
EB53: AC 85 FE           LDY    VCBOFFS      ;index current VCB
EB56: B9 10 D9           LDA    VCUNUM,Y     ;get new VBM unit number
EB59: 8D 9A FE           STA    VBMDVNUM     ;save new VBM unit number
EB5C: AC 99 FE  :1       LDY    VBMFLG
EB5F: 30 05              BMI    :2           ;bitmap block already changed
EB61: 20 87 EB           JSR    DORDVBM      ;read VBM block
EB64: B0 0B              BCS    RTS10        ;error
EB66: AC 85 FE  :2       LDY    VCBOFFS      ;index current VCB
EB69: B9 1C D9           LDA    VCNXTVBM,Y   ;get block offset into VBM
EB6C: 0A                 ASL                 ;double it
EB6D: 8D 97 FE           STA    VBMPGOFS     ;save page offset into VBM
EB70: 18                 CLC                 ;signal no error
EB71: 60        RTS10    RTS
                *-----------------------------
                * SET DISK FULL ERROR:
                *-----------------------------
EB72: A9 48     DFULLERR LDA    #$48         ;disk full error code
```

LISTING 1: DESTROY: *(continued)*

```
EB74: 38                     SEC                ;signal error
EB75: 60                     RTS
                    *---------------------------------
                    * CHECKPOINT VOLUME BITMAP FOR DISK WRITING:
                    *---------------------------------
EB76: 18            CKPTVBM  CLC                ;assume no error
EB77: AD 99 FE               LDA  VBMFLG        ;get VBM write-needed flag
EB7A: 10 F5                  BPL  RTS10         ;write not necessary
EB7C: 20 D1 EB               JSR  WRVBMBLK      ;write VBM block to disk
EB7F: B0 F0                  BCS  RTS10         ;write error
EB81: A9 00                  LDA  #0
EB83: 8D 99 FE               STA  VBMFLG        ;clear write-needed flag
EB86: 60                     RTS
                    *---------------------------------
                    * PREPARE TO READ VOLUME BITMAP BLOCK:
                    *---------------------------------
EB87: 8D 9A FE      DORDVBM  STA  VBMDVNUM      ;save device code
EB8A: AC 85 FE               LDY  VCBOFFS       ;index current VCB
EB8D: B9 1C D9               LDA  VCNXTVBM,Y    ;get next VBM block from VCB
EB90: 8D 9D FE               STA  VBMBLOFS      ; and save in work area
EB93: 18                     CLC                ;VBM+VCB block offsets=
EB94: 79 1A D9               ADC  VCVBMOFS,Y    ; VBM block to get
EB97: 8D 9B FE               STA  VBMBLKNM      ;save block number (LSB)
EB9A: B9 1B D9               LDA  VCVBMOFS+1,Y
EB9D: 69 00                  ADC  #0            ;pick up carry
EB9F: 8D 9C FE               STA  VBMBLKNM+1    ;save block number (MSB)
EBA2: A9 01                  LDA  #1            ;set read command
                    *---------------------------------
                    * READ/WRITE VOLUME BITMAP BLOCK:
                    *---------------------------------
EBA4: 85 42         RWVBMBLK STA  DDCMDNUM      ;save ZP command number
EBA6: AD 30 BF               LDA  DEVNUM
EBA9: 48                     PHA                ;save entry device code on stack
EBAA: AD 9A FE               LDA  VBMDVNUM
EBAD: 8D 30 BF               STA  DEVNUM        ;set new device number
EBB0: AD 9B FE               LDA  VBMBLKNM
EBB3: 85 46                  STA  DDBLKNUM      ;set ZP block number (LSB)
EBB5: AD 9C FE               LDA  VBMBLKNM+1
EBB8: 85 47                  STA  DDBLKNUM+1    ;set ZP block number (MSB)
EBBA: AD 82 EA               LDA  FRVBM2+2      ;point to VBM buffer
EBBD: 20 DF EB               JSR  DOIO          ;read the block
EBC0: AA                     TAX                ;save error code
EBC1: 68                     PLA
EBC2: 8D 30 BF               STA  DEVNUM        ;restore entry device code
EBC5: 90 01                  BCC  :1            ;no error
EBC7: 8A                     TXA                ;read error, restore error code
EBC8: 60            :1       RTS
                    *---------------------------------
                    * READ BLOCK NUMBER IN A,X REGISTERS:
                    *---------------------------------
EBC9: 85 46         RDBLKAX  STA  DDBLKNUM
EBCB: 86 47                  STX  DDBLKNUM+1
EBCD: 20 D9 EB               JSR  READBLK
EBD0: 60                     RTS
                    *---------------------------------
                    * WRITE VOLUME BITMAP BLOCK:
                    *---------------------------------
EBD1: A9 02         WRVBMBLK LDA  #2            ;set write code
EBD3: D0 CF                  BNE  RWVBMBLK      ;always
                    *---------------------------------
                    * WRITE PRIMARY BLOCK BUFFER BLOCK:
                    *---------------------------------
EBD5: A9 02         WRITBLK  LDA  #2            ;set write code
EBD7: D0 02                  BNE  RWBLK         ;always
                    *---------------------------------
                    * READ PRIMARY BLOCK BUFFER BLOCK:
                    *---------------------------------
EBD9: A9 01         READBLK  LDA  #1            ;set read code
                    *---------------------------------
                    * READ/WRITE PRIMARY BLOCK BUFFER BLOCK:
                    *---------------------------------
EBDB: 85 42         RWBLK    STA  DDCMDNUM      ;save command number
EBDD: A9 DC                  LDA  #>PBLKBUF     ;point to Primary Block Buffer
                    *---------------------------------
                    * READ/WRITE BLOCK:
                    *---------------------------------
EBDF: 08            DOIO     PHP                ;save entry status reg
EBE0: 78                     SEI                ;disable interrupts for I/O
EBE1: 85 45                  STA  DDBUFPTR+1    ;save I/O buffer (MSB)
EBE3: A9 00                  LDA  #0            ;always zero
EBE5: 85 44                  STA  DDBUFPTR      ; I/O buffer (LSB)
EBE7: 8D 0F BF               STA  SERR          ;zero global page error location
EBEA: A9 FF                  LDA  #$FF          ;indicate that
EBEC: 8D A6 FE               STA  IOTFRFLG      ; I/O occurred
EBEF: AD 30 BF               LDA  DEVNUM
EBF2: 85 43                  STA  DDSLTDRV      ;set ZP device code
EBF4: 20 E4 DE               JSR  TOBLKIO       ;do I/O
EBF7: B0 03                  BCS  :1            ;I/O error
EBF9: 28                     PLP                ;restore entry status reg
EBFA: 18                     CLC                ;signal no error
EBFB: 60                     RTS
EBFC: 28            :1       PLP                ;restore entry status reg
EBFD: 38                     SEC                ;signal error
EBFE: 60                     RTS
                    *=================================
                    * MLI DESTROY COMMAND HANDLER:
                    *=================================
                             ORG  $F932

F932: 20 93 E5      MLIDSTRY JSR  GFILENT       ;get file entry
F935: B0 47                  BCS  SECRTS1       ;error
F937: 20 9B EF               JSR  GETFCB        ;get FCB
F93A: AD 88 FE               LDA  FCBOPFLG
F93D: D0 3D                  BNE  :3            ;file open error
F93F: A9 00                  LDA  #0
F941: 8D 88 FE               STA  FBLKNEED      ;indicate no free
F944: 8D 89 FE               STA  FBLKNEED+1    ; blocks needed
F947: 20 59 E9               JSR  GFREBLK       ;calculate VCB free block count
F94A: 90 04                  BCC  :1            ;no error
F94C: C9 48                  CMP  #$48          ;disk device full error code?
F94E: D0 2E                  BNE  SECRTS1       ;no, fatal error
F950: AD 71 FE      :1       LDA  ACCESS        ;check file access attribute
F953: 29 80                  AND  #$80
F955: D0 05                  BNE  :2            ;destroy bit enabled
F957: A9 4E                  LDA  #$4E          ;access error code
F959: 20 09 BF               JSR  SYSERR        ;handle error
F95C: AD 30 BF      :2       LDA  DEVNUM        ;check device
F95F: 20 3E F4               JSR  STATCALL      ; status
F962: B0 1A                  BCS  SECRTS1       ;device status error
F964: AD 64 FE               LDA  FEKEYPTR      ;copy key block number
F967: 8D B3 FE               STA  KYBLKPTR      ; of file from File
F96A: AD 65 FE               LDA  FEKEYPTR+1    ; Entry Buffer to
F96D: 8D B4 FE               STA  KYBLKPTR+1    ; Variable Work Area
F970: AD 53 FE               LDA  FESTYPNL      ;get storage.type/name.length
F973: 29 F0                  AND  #$F0          ;isolate storage.type*16
F975: C9 40                  CMP  #$40
F977: 90 07                  BCC  DSTRYFIL      ;nondirectory file
F979: 4C DE F9               JMP  DSTRYDIR      ;directory file
F97C: A9 50         :3       LDA  #$50          ;file open error code
F97E: 38            SECRTS1  SEC                ;signal error
F97F: 60                     RTS
                    *---------------------------------
                    * DESTROY NON-DIRECTORY FILE:
                    *---------------------------------
F980: 8D B5 FE      DSTRYFIL STA  STORTYP       ;save storage.type*16
F983: A2 05                  LDX  #5            ;index 5 bytes after STORTYP
F985: A9 00                  LDA  #0            ;zero the 5 bytes
F987: 9D B5 FE      :1       STA  STORTYP,X     ;zero: VWFREBLK (LSB,MSB)
F98A: CA                     DEX                ;      EOFBLKNM (LSB,MSB)
F98B: D0 FA                  BNE  :1            ;      EOFBLKOF (LSB)
F98D: A9 02                  LDA  #2            ;set $200 bytes as
F98F: 8D BB FE               STA  EOFBLKOF+1    ; the byte offset
F992: EE EB FE               INC  DSTRYFLG      ;set the destroy flag
F995: 20 44 FA               JSR  TRUNCEOF      ;truncate file at EOF
F998: CE EB FE               DEC  DSTRYFLG      ;clear the destroy flag
F99B: B0 E1                  BCS  SECRTS1       ;truncation error
F99D: AE B4 FE      DSTRYFI1 LDX  KYBLKPTR+1    ;designate key block as
F9A0: AD B3 FE               LDA  KYBLKPTR      ; block to be freed
F9A3: 20 1A EA               JSR  FRVBMBLK      ;free key block in VBM
F9A6: B0 D6                  BCS  SECRTS1       ;error
F9A8: A9 00                  LDA  #0            ;zero storage.type/name.length
F9AA: 8D 53 FE               STA  FESTYPNL      ; to indicate file deletion
F9AD: CD 47 FE               CMP  VDFILCNT      ;decrement file
F9B0: D0 03                  BNE  :1            ; count in
F9B2: CE 48 FE               DEC  VDFILCNT+1    ; Variable
F9B5: CE 47 FE      :1       DEC  VDFILCNT      ; Data Area
F9B8: 20 76 EB               JSR  CKPTVBM       ;checkpoint VBM
F9BB: B0 C1                  BCS  SECRTS1       ;ckeckpoint error
F9BD: 20 C3 F9               JSR  UPDVCBFR      ;update free block count in VCB
F9C0: 4C B2 E4               JMP  UPDATDIR      ;update directory
                    *---------------------------------
                    * UPDATE FREE BLOCK COUNT IN VCB:
                    *---------------------------------
F9C3: AC 85 FE      UPDVCBFR LDY  VCBOFFS       ;get file index into FCB
F9C6: AD B6 FE               LDA  VWFREBLK      ;add blocks freed to
F9C9: 79 14 D9               ADC  VCFREBLK,Y    ; total free blocks
F9CC: 99 14 D9               STA  VCFREBLK,Y
F9CF: AD B7 FE               LDA  VWFREBLK+1
F9D2: 79 15 D9               ADC  VCFREBLK+1,Y
F9D5: 99 15 D9               STA  VCFREBLK+1,Y
F9D8: A9 00                  LDA  #0            ;start next search for free
F9DA: 99 1C D9               STA  VCNXTVBM,Y    ; blocks at beginning of VBM
F9DD: 60                     RTS
                    *---------------------------------
                    * DESTROY DIRECTORY FILE:
                    *---------------------------------
F9DE: C9 D0         DSTRYDIR CMP  #$D0          ;subdirectory file code*16
F9E0: D0 4B                  BNE  :6            ;not subdirectory file
F9E2: 20 43 EB               JSR  RDVBMBLK      ;read VBM block
F9E5: B0 45                  BCS  :5            ;read error
F9E7: AD 64 FE               LDA  FEKEYPTR      ;copy key block pointer
F9EA: 85 46                  STA  DDBLKNUM      ; from File Entry Buffer
F9EC: AD 65 FE               LDA  FEKEYPTR+1    ; into ZP
F9EF: 85 47                  STA  DDBLKNUM+1    ; block number location
F9F1: 20 D9 EB               JSR  READBLK       ;read key block
F9F4: B0 36                  BCS  :5            ;read error
F9F6: AD 25 DC               LDA  PBFILCNT
F9F9: D0 05                  BNE  :1            ;directory not empty
F9FB: AD 26 DC               LDA  PBFILCNT+1
F9FE: F0 05                  BEQ  :2            ;directory empty
FA00: A9 4E         :1       LDA  #$4E          ;access error code
FA02: 20 09 BF               JSR  SYSERR        ;handle error
FA05: 8D 04 DC      :2       STA  PBSTYPNL      ;zero storage.type/name.length
FA08: 20 D5 EB               JSR  WRITBLK       ;write key block back to disk
FA0B: B0 1F                  BCS  :5            ;write error
FA0D: AD 02 DC      :3       LDA  PBFWDPTR      ;forward link (LSB)
FA10: CD 03 DC               CMP  PBFWDPTR+1
FA13: D0 04                  BNE  :4            ;more file blocks to free in VBM
FA15: C9 00                  CMP  #0
FA17: F0 84                  BEQ  DSTRYFI1      ;no more file blocks to free
FA19: AE 03 DC      :4       LDX  PBFWDPTR+1    ;forward link (MSB)
FA1C: 20 1A EA               JSR  FRVBMBLK      ;free block in VBM
FA1F: B0 0B                  BCS  :5            ;error
FA21: AD 02 DC               LDA  PBFWDPTR      ;get next file
FA24: AE 03 DC               LDX  PBFWDPTR+1    ; block to free
FA27: 20 C9 EB               JSR  RDBLKAX       ;read block
FA2A: 90 E1                  BCC  :3            ;loop back 'til all blocks freed
FA2C: 60            :5       RTS
FA2D: A9 4A         :6       LDA  #$4A          ;incompatible file format code
FA2F: 20 09 BF               JSR  SYSERR        ;handle error
                    *---------------------------------
                    * SET WRITE-OCCURRED FLAG:
                    *---------------------------------
FA32: 48                     PHA                ;save entry A-reg
FA33: 98                     TYA
FA34: 48                     PHA                ;save entry Y-reg
FA35: AC 86 FE               LDY  FCBOFFS       ;get file index into FCB
FA38: B9 1C D8               LDA  WRITFLG,Y
FA3B: 09 80                  ORA  #$80          ;turning on bit 7 sets
FA3D: 99 1C D8               STA  WRITFLG,Y     ; write-occurred flag
FA40: 68                     PLA
FA41: A8                     TAY                ;restore entry Y-reg
FA42: 68                     PLA                ;restore entry A-reg
```

LISTING 1: DESTROY: *(continued)*

```
FA43: 60                     RTS
                    *--------------------------------
                    * TRUNCATE FILE AT EOF:
                    *--------------------------------
FA44: AD B5 FE      TRUNCEOF LDA   STORTYP      ;get storage type*16
FA47: C9 20                  CMP   #$20
FA49: 90 0D                  BCC   :1           ;seedling file
FA4B: C9 30                  CMP   #$30
FA4D: 90 0C                  BCC   :2           ;sapling file
FA4F: C9 40                  CMP   #$40
FA51: 90 0B                  BCC   TRNCTREE     ;tree file
FA53: A9 0C                  LDA   #$0C         ;death code
FA55: 20 0C BF               JSR   SYSDEATH     ;arrgh!
FA58: 4C 2F FB      :1       JMP   TRNCSEED     ;go to seedling truncate
FA5B: 4C F6 FA      :2       JMP   TRNCSAP      ;go to sapling truncate
                    *--------------------------------
                    * TRUNCATE OR DELETE TREE FILE:
                    *--------------------------------
                    * Alter entries in subindex blocks:

FA5E: A9 80         TRNCTREE LDA   #128         ;up to 128 subindex block
FA60: 8D BC FE               STA   EOFMIX       ; numbers in master index block
FA63: 20 5A FB      :1       JSR   RDKEYBLK     ;read master index block
FA66: B0 60                  BCS   RTS1         ;read error
FA68: AC BC FE               LDY   EOFMIX       ;Y=master index block EOF
FA6B: CC B8 FE               CPY   EOFBLKNM
FA6E: F0 59                  BEQ   TRNCTRE1     ;at EOF in master index block
FA70: A2 07                  LDX   #7           ;handle up to 8 subindex blocks
FA72: B9 00 DC      :2       LDA   PBLKBUF,Y    ;copy subindex block entry
FA75: 9D BE FE               STA   DEVTBL,X     ; (LSB) to Device Table
FA78: 19 00 DD               ORA   PBLKBUF+256,Y
FA7B: F0 09                  BEQ   :3           ;zero entry found
FA7D: B9 00 DD               LDA   PBLKBUF+256,Y ;copy subindex block entry
FA80: 9D C6 FE               STA   DEVTBL+8,X   ; (MSB) to table
FA83: CA                     DEX                ;reduce counter
FA84: 30 12                  BMI   :5           ;8 block numbers copied
FA86: 88            :3       DEY                ;index next lower subindex entry
FA87: CC B8 FE               CPY   EOFBLKNM
FA8A: D0 E6                  BNE   :2           ;not at EOF so loop back
FA8C: C8                     INY                ;at EOF so fill remainder of
FA8D: A9 00                  LDA   #0           ; Device Table with zeros
FA8F: 9D BE FE      :4       STA   DEVTBL,X
FA92: 9D C6 FE               STA   DEVTBL+8,X
FA95: CA                     DEX
FA96: 10 F7                  BPL   :4           ;loop back until table full
FA98: 88            :5       DEY                ;save index to next
FA99: 8C BC FE               STY   EOFMIX       ; subindex block entry
FA9C: A2 07                  LDX   #7           ;handle up to 8 subindex blocks
FA9E: 8E BD FE      :6       STX   DEVTBLIX     ;save index to subindex table
FAA1: BD BE FE               LDA   DEVTBL,X     ;copy subindex block entry (LSB)
FAA4: 85 46                  STA   DDBLKNUM     ; to ZP block number (LSB)
FAA6: 1D C6 FE               ORA   DEVTBL+8,X
FAA9: F0 B8                  BEQ   :1           ;zero entry found so exit
FAAB: BD C6 FE               LDA   DEVTBL+8,X   ;copy subindex block entry
FAAE: 85 47                  STA   DDBLKNUM+1   ; (MSB) to ZP block number (MSB)
FAB0: 20 D9 EB               JSR   READBLK      ;read subindex block
FAB3: B0 13                  BCS   RTS1         ;read error
FAB5: 20 95 FB               JSR   FREIXBLK     ;free subindex block
FAB8: B0 0E                  BCS   RTS1         ;error
FABA: 20 D5 EB               JSR   WRITBLK      ;write altered subindex block
FABD: B0 09                  BCS   RTS1         ;write error
FABF: AE BD FE               LDX   DEVTBLIX     ;restore index to subindex table
FAC2: CA                     DEX                ;reduce index
FAC3: 10 D9                  BPL   :6           ;loop back until table completed
FAC5: 30 9C                  BMI   :1           ;get master index block again
FAC7: 18            CLCRTS1  CLC                ;signal no error
FAC8: 60            RTS1     RTS

                    * Alter entries after EOF in master index block:

FAC9: AC B8 FE      TRNCTRE1 LDY   EOFBLKNM     ;start at one entry beyond EOF
FACC: C8                     INY                ; in master index block
FACD: 20 97 FB               JSR   FREIXBL1     ;free all entries beyond EOF
FAD0: B0 F6                  BCS   RTS1         ;error
FAD2: 20 D5 EB               JSR   WRITBLK      ;write master block back to disk
FAD5: B0 F1                  BCS   RTS1         ;write error
FAD7: AC B8 FE               LDY   EOFBLKNM     ;if EOF in 1st subindex block
FADA: F0 15                  BEQ   :1           ; then demote tree to sapling
FADC: B9 00 DC               LDA   PBLKBUF,Y    ;get subindex block number (LSB)
FADF: 85 46                  STA   DDBLKNUM     ; which contains EOF
FAE1: 19 00 DD               ORA   PBLKBUF+256,Y
FAE4: F0 E1                  BEQ   CLCRTS1      ;none found
FAE6: B9 00 DD               LDA   PBLKBUF+256,Y ;get subindex block number
FAE9: 85 47                  STA   DDBLKNUM+1   ; (MSB) which contains EOF
FAEB: 20 D9 EB               JSR   READBLK      ;read final subindex block and
FAEE: 90 0B                  BCC   TRNCSAP1     ; treat it as sapling file
FAF0: 60                     RTS
FAF1: 20 63 FB      :1       JSR   DEMFITYP     ;demote tree to sapling file
FAF4: B0 D2                  BCS   RTS1         ;error
                    *--------------------------------
                    * TRUNCATE OR DELETE SAPLING FILE:
                    *--------------------------------
FAF6: 20 5A FB      TRNCSAP  JSR   RDKEYBLK     ;read key index block
FAF9: B0 CD                  BCS   RTS1         ;read error
FAFB: AC B9 FE      TRNCSAP1 LDY   EOFBLKNM+1   ;start at one entry beyond EOF
FAFE: C8                     INY                ; in index block
FAFF: F0 0A                  BEQ   :1           ;no blocks to free
FB01: 20 97 FB               JSR   FREIXBL1     ;free all entries beyond EOF
FB04: B0 C2                  BCS   RTS1         ;error
FB06: 20 D5 EB               JSR   WRITBLK      ;write index block back to disk
FB09: B0 BD                  BCS   RTS1         ;write error
FB0B: AC B9 FE      :1       LDY   EOFBLKNM+1   ;get last index block in file
FB0E: F0 15                  BEQ   :3           ;last index block empty
FB10: B9 00 DC      :2       LDA   PBLKBUF,Y    ;get last data block number
FB13: 85 46                  STA   DDBLKNUM     ; (LSB) in file
FB15: 19 00 DD               ORA   PBLKBUF+256,Y
FB18: F0 AD                  BEQ   CLCRTS1      ;none found
FB1A: B9 00 DD               LDA   PBLKBUF+256,Y ;get last data block number
FB1D: 85 47                  STA   DDBLKNUM+1   ; (MSB) in file
FB1F: 20 D9 EB               JSR   READBLK      ;read last data block
FB22: 90 10                  BCC   TRNCSEE1     ; and handle it as seedling
FB24: 60                     RTS
FB25: AD B8 FE      :3       LDA   EOFBLKNM     ;if more index blocks,
FB28: D0 E6                  BNE   :2           ; then file is tree,
FB2A: 20 63 FB               JSR   DEMFITYP     ; else demote to seedling
FB2D: B0 2A                  BCS   RTS2         ;error
                    *--------------------------------
                    * TRUNCATE OR DELETE SEEDLING FILE:
                    *--------------------------------
FB2F: 20 5A FB      TRNCSEED JSR   RDKEYBLK     ;read key data block
FB32: B0 25                  BCS   RTS2         ;read error
FB34: AC BB FE      TRNCSEE1 LDY   EOFBLKOF+1   ;get EOF page
FB37: F0 06                  BEQ   :1           ;EOF in 1st page
FB39: 88                     DEY                ;reduce offset
FB3A: D0 1C                  BNE   :5           ;EOF on page boundary
FB3C: AC BA FE               LDY   EOFBLKOF     ;get EOF offset in 2nd page
FB3F: A9 00         :1       LDA   #0           ;zero required bytes in 2nd page
FB41: 99 00 DD      :2       STA   PBLKBUF+256,Y
FB44: C8                     INY
FB45: D0 FA                  BNE   :2           ;loop back until done
FB47: AC BB FE               LDY   EOFBLKOF+1   ;get EOF page
FB4A: D0 09                  BNE   :4           ;EOF in page 2 so skip 1st page
FB4C: AC BA FE               LDY   EOFBLKOF     ;get EOF offset in 1st page
FB4F: 99 00 DC      :3       STA   PBLKBUF,Y    ;zero required bytes in 1st page
FB52: C8                     INY
FB53: D0 FA                  BNE   :3           ;loop back until done
FB55: 4C D5 EB      :4       JMP   WRITBLK      ;write block back to disk
FB58: 18            :5       CLC                ;signal no error
FB59: 60            RTS2     RTS
                    *--------------------------------
                    * READ KEY BLOCK:
                    *--------------------------------
FB5A: AD B3 FE      RDKEYBLK LDA   KYBLKPTR
FB5D: AE B4 FE               LDX   KYBLKPTR+1
FB60: 4C C9 EB               JMP   RDBLKAX      ;read block in A,X-regs
                    *--------------------------------
                    * DEMOTE FILE TYPE:
                    *--------------------------------
FB63: AE B4 FE      DEMFITYP LDX   KYBLKPTR+1   ;get key index block number into
FB66: 8A                     TXA                ; A,X-regs and save on stack
FB67: 48                     PHA
FB68: AD B3 FE               LDA   KYBLKPTR
FB6B: 48                     PHA
FB6C: 20 1A EA               JSR   FRVBMBLK     ;free block in VBM
FB6F: 68                     PLA                ;restore key index block
FB70: 85 46                  STA   DDBLKNUM     ; number from stack and
FB72: 68                     PLA                ; stuff in ZP block number
FB73: 85 47                  STA   DDBLKNUM+1
FB75: B0 1D                  BCS   :1           ;error
FB77: AD 00 DC               LDA   PBLKBUF      ;1st index block in old key
FB7A: 8D B3 FE               STA   KYBLKPTR     ; block becomes new key block
FB7D: AD 00 DD               LDA   PBLKBUF+256
FB80: 8D B4 FE               STA   KYBLKPTR+1
FB83: A0 00                  LDY   #0           ;alter 1st entry in
FB85: 20 C7 FB               JSR   DOIXBYT      ; old key index block
FB88: 38                     SEC
FB89: AD B5 FE               LDA   STORTYP      ;get old storage type and
FB8C: E9 10                  SBC   #$10         ; reduce it to reflect
FB8E: 8D B5 FE               STA   STORTYP      ; demoted storage type
FB91: 20 D5 EB               JSR   WRITBLK      ;write block back to disk
FB94: 60            :1       RTS
                    *--------------------------------
                    * FREE INDEX BLOCK ENTRIES:
                    *--------------------------------
FB95: A0 00         FREIXBLK LDY   #0           ;enter here to free entire block
FB97: A5 46         FREIXBL1 LDA   DDBLKNUM     ; and here to free partial block
FB99: 48                     PHA                ;save block number on stack
FB9A: A5 47                  LDA   DDBLKNUM+1
FB9C: 48                     PHA
FB9D: 8C 91 FE      :1       STY   YSAV         ;save index to index block
FBA0: B9 00 DC               LDA   PBLKBUF,Y    ;get block number (LSB)
FBA3: D9 00 DD               CMP   PBLKBUF+256,Y
FBA6: D0 04                  BNE   :2           ;nonzero entry for processing
FBA8: C9 00                  CMP   #0
FBAA: F0 0E                  BEQ   :3           ;skip zero entry
FBAC: BE 00 DD      :2       LDX   PBLKBUF+256,Y ;get block number (MSB)
FBAF: 20 1A EA               JSR   FRVBMBLK     ;free block in VBM
FBB2: B0 0A                  BCS   :4           ;error
FBB4: AC 91 FE               LDY   YSAV         ;restore index to index block
FBB7: 20 C7 FB               JSR   DOIXBYT      ;alter index block
FBBA: C8            :3       INY
FBBB: D0 E0                  BNE   :1
FBBD: 18                     CLC                ;signal no error
FBBE: AA            :4       TAX                ;save possible error code
FBBF: 68                     PLA                ;restore block number from stack
FBC0: 85 47                  STA   DDBLKNUM+1
FBC2: 68                     PLA
FBC3: 85 46                  STA   DDBLKNUM
FBC5: 8A                     TXA                ;restore possible error code
FBC6: 60                     RTS
                    *--------------------------------
                    * ZERO OR SWAP BYTES IN INDEX BLOCK:
                    *--------------------------------
FBC7: AD EB FE      DOIXBYT  LDA   DSTRYFLG     ;get value of destroy flag
FBCA: D0 03                  BNE   :1           ;destroying means swapping
FBCC: AA                     TAX                ;truncating means zeroing
FBCD: 80 06                  BRA   :2           ;you devil you! (BEQ in v1.4)
FBCF: BE 00 DD      :1       LDX   PBLKBUF+256,Y ;prepare to invert
FBD2: B9 00 DC               LDA   PBLKBUF,Y    ; index block entry
FBD5: 99 00 DD      :2       STA   PBLKBUF+256,Y ;set new MSB
FBD8: 8A                     TXA
FBD9: 99 00 DC               STA   PBLKBUF,Y    ;set new LSB
FBDC: 60                     RTS
                    *--------------------------------
                    * VOLUME BITMAP BYTE MASK TABLE:
                    *--------------------------------
                             ORG   $FDF4

FDF4: 80 40 20      VBMSKTBL HEX   80,40,20,10,08,04,02,01
FDF7: 10 08 04 02 01

--End assembly, 1181 bytes, Errors: 0
```

END OF LISTING 1