Apple II
Technical Notes

Developer Technical Support

ProDOS 8
#30: Sparse Station

Written by: Matt Deatherage May 1992

This Technical Note discusses issues when using sparse files under ProDOS 8.

Sparse Information Available

The concept of sparse files is introduced in the *ProDOS 8 Technical Reference Manual* in sometimes confusing language. The concept behind sparse files is pretty simple. If you didn't think it could be explained in two paragraphs, have a seat and learn something.

The ProDOS file system keeps track of where files reside on disk through a series of "index blocks." All index blocks are disk blocks that contain lists of block numbers. They may be organized in several ways (seedling, sapling or tree), depending on how big the file is—one 512-byte block can hold 256 two-byte block numbers. If a file is one block long, it has no index blocks and is a **seedling** file. If a non-sparse file is between two and 256 blocks long, it has one index block and is a **sapling** file. If a non-sparse file is longer than 256 blocks, it's a **tree** file and has a "master index block" that points to other index blocks. This is more than enough to store any ProDOS file—one master index block pointing to 256 other index blocks, each of which points to 256 data blocks on disk would be a 32 MB file—twice the limit of 16 MB imposed by ProDOS's 3-byte storage for file lengths.

What happens if you don't need to use all of those blocks? For example, if you need to store data at file offset $0000 and at file offset $20000, does ProDOS make you waste 256 disk blocks you're not going to use? Fortunately, the answer is "no." ProDOS lets you skip any data block you're not using by recording a pointer to data block $0000 instead of to a regular block on the disk. When ProDOS sees a block pointer of $0000 in an index block, it knows not to read block zero (which contains boot code) but instead to pretend that it read a block of zeroes from the disk. This lets you save lots of space on disk—a file created this way is a **sparse** file. (See? Two paragraphs.)

Under ProDOS 8, you can create a sparse file by using the `SET_EOF` MLI command to extend the file's current end-of-file position, and then using `SET_MARK` to move the mark to the new end-of-file position. If you grow a file by increasing the EOF but not actually writing data, ProDOS 8 makes the blocks you skip sparse. Under GS/OS, the ProDOS FST automatically converts long stretches of zeroes to sparse blocks, making sparse files even more prevalent.

Dealing With Sparsity

Unfortunately, ProDOS 8 does **not** automatically make sparse files when you write large sections of zeroes. That means if you read a sparse file and write it back out, you "expand" it and it's no longer sparse. The file could balloon to hundreds of times its previous disk space, which is not a good thing.

So how do you recognize a sparse file? You can notice that the length of the file has to be pretty close to 512 bytes multiplied by the number of blocks allocated to data in the file. For example, take a file that's $4068 bytes long. $4068 bytes takes 33 512-byte blocks—32 blocks is $4000 bytes, plus one more block for the last $68 bytes. This is between 2 and 256 blocks, so there's one more block allocated for the index block. If this file is not sparse, it uses 34 blocks on disk. If it uses any **less** than 34 blocks in reality, it's sparse.

This calculation gets a little tricker for tree files—if the file has more than 256 data blocks, add one master index block plus one index block for each 256 data blocks or portion thereof. To give another example, a file that's $68D3F bytes long takes 839 ($347) data blocks. This file has five additional blocks allocated to it—one master index block and four index blocks. The first three index blocks are full (256 × 3 = 768) and the fourth contains the remaining 71 data blocks. If this file takes less than 844 blocks on disk, it's sparse.

Too Complicated?

For all except very speedy utilities to copy files, yes. If you just need an easy way to deal with sparse files that's not so speed-critical, read on.

All you have to do to preserve (or create) sparsity in normal file copying operations is scan the data you've read from disk before you write it back. Suppose your file copying buffer is 10K large. Read 10K of data from your source file, then divide the buffer into 512-byte chunks and scan the data looking for zeroes. If you find a non-zero byte, write the entire 512-byte chunk of data to the target file and proceed to the next 512-byte chunk. If you don't find any non-zero bytes in a 512-byte chunk, just set the mark ahead 512 bytes and don't issue a `WRITE` call. This is basically how GS/OS's ProDOS FST automatically sparses files, and it can work for you too.

Is It That Easy?

Well, no. There's an important exception—AppleShare.

Most AppleShare servers (including all of Apple's) don't support sparse files—all the logical blocks you use have to be physically allocated on the server's hard disk. The following BASIC.SYSTEM command:

```
BSAVE SPARSE.FILE,A$300,L$1,B$FFFFFF
```

creates a 16 MB sparse file with one byte of logical data in it. This file only takes 5 blocks on a ProDOS disk (one master index block, two index blocks and two data blocks—it takes two data blocks because ProDOS 8 always allocates the very first block of a file when you create it, even if you don't use the first 512 bytes), but it takes 16 MB of disk space on a server.

That's not all—for speed reasons, AppleShare does **not** fill the extra, normally-sparsed blocks with zeroes. If you issued the above command to an AppleShare server under ProDOS 8 and then tried to read the first few bytes of the resulting file, they would be garbage—but not zeroes.

ProDOS 8 Technical Note #21 gives information on identifying AppleShare server volumes—if you're dealing with one, do **not** use normal sparse file creation techniques. Just write the 512 bytes of zeroes instead of advancing the mark. It doesn't take any more disk space and it achieves the results you want.

One More Thing

In versions of ProDOS 8 up to 1.9, setting the end-of-file position past $200 on a **seedling** file created a sparse file that confused ProDOS 8 if you ever used `SET_EOF` on it again. This is fixed in version 2.0.1 and later.

Further Reference

- *ProDOS 8 Technical Reference Manual*