Apple II
Technical Notes

Developer Technical Support

ProDOS 8
#25: Non-Standard Storage Types

Revised by: Matt Deatherage December 1991
Written by: Matt Deatherage July 1989

This Technical Note discusses storage types for ProDOS files which are not documented in the *ProDOS 8 Technical Reference Manual.*

Warning: The information provided in this Note is for the use of disk utility programs which occasionally must manipulate non-standard files in unusual situations. ProDOS 8 programs should not create or otherwise manipulate files with non-standard storage types.

Changes since July 1989: Included new information on storing HFS Finder information in extended files' extended key blocks.

Introduction

One of the features of the ProDOS file system is its ability to let ProDOS 8 know when someone has put a file on the disk that ProDOS 8 can't access. A file not created by ProDOS 8 can be identified by the `storage_type` field. ProDOS 8 creates four different storage types: seedling files ($1), sapling files ($2), tree files ($3), and directory files ($D). ProDOS 8 also stores subdirectory headers as storage type $E and volume directory headers as storage type $F. These are all described in the *ProDOS 8 Technical Reference Manual.*

Other files may be placed on the disk, and ProDOS 8 can catalog them, rename them, and return file information about them. However, since it does not know how the information in the files is stored on the disk, it cannot perform normal file operations on these files, and it returns the `Unsupported Storage Type` error instead.

Apple reserves the right to define additional storage types for the extension of the ProDOS file system in the future. To date, two additional storage types have been defined. Storage type $4 indicates a Pascal area on a ProFile hard disk, and storage type $5 indicates a GS/OS extended file (data fork and resource fork) as created by the ProDOS FST.

Storage Type $4

Storage type $4 is used for Apple II Pascal areas on Profile hard disk drives. These files are created by the Apple Pascal ProFile Manager. Other programs should not create these files, as Apple II Pascal could freak out.

The Pascal Profile Manager (PPM) creates files which are internally divided into pseudo-volumes by Apple II Pascal. The files have the name PASCAL.AREA (name length of 10), with file type $EF. The `key_pointer` field of the directory entry points to the first block used by the file, which is the second to last block on the disk. As ProDOS stores files non-contiguously up from the bottom, PPM creates pseudo-volumes contiguously down from the end of the ProFile. `Blocks_used` is 2, and `header_pointer` is also 2. All other fields in the directory are set to 0. PPM looks for this entry (starting with the name PASCAL.AREA) to determine if a ProFile has been initialized for Pascal use.

The file entry for the Pascal area increments the number of files in the ProDOS directory and the `key_pointer` for the file points to `TOTAL_BLOCKS` - 2, or the second to last block on the disk. When PPM expands or contracts the Pascal area, `blocks_used` and `key_pointer` are updated accordingly. With any access to this entry (such as adding or deleting pseudo-volumes within PPM), the backup bit is not set (PPM provides a utility to back up the Pascal area).

The Pascal volume directory contains two separate contiguous data structures that specify the contents of the Pascal area on the Profile. The volume directory occupies two blocks to support 31 pseudo-volumes. It is found at the physical block specified in the ProDOS volume directory as the value of `key_pointer` (i.e., it occupies the first block in the area pointed to by this value).

The first portion of the volume directory is the actual directory for the pseudo-volumes. It is an array with the following Apple II Pascal declaration:

```
TYPE   RTYPE = (HEADER, REGULAR)

VAR    VDIR:  ARRAY [0..31] OF
                    PACKED RECORD
                          CASE RTYPE OF
                                HEADER:      (PSEUDO_DEVICE_LENGTH:INTEGER;
                                              CUR_NUM_VOLS:INTEGER;
                                              PPM_NAME:STRING[3]);
                                REGULAR:     (START:INTEGER;
                                              DEFAULT_UNIT:0.255
                                              FILLER:0..127
                                              WP:BOOLEAN
                                              OLDDRIVERADDR:INTEGER
                    END;
```

The `HEADER` specifies information about the Pascal area. It specifies the size in blocks in `PSEUDO_DEVICE_LENGTH`, the number of currently allocated volumes in `CUR_NUM_VOLS`, and a special validity check in `PPM_NAME`, which is the three-character string PPM. The header information is accessed via a reference to `VDIR[0]`. The `REGULAR` entry specifies information for each pseudo-volume. `START` is the starting block address for the pseudo-volume, and `LENGTH` is the length of the pseudo-volume in blocks. `DEFAULT_UNIT` specifies the default

Pascal unit number that this pseudo-volume should be assigned to upon booting the system. This value is set through the Volume Manager by either the user or an application program, and it remains valid if it is not released.

If the system is shut down, the pseudo-volume remains assigned and will be active once the system is rebooted. `WP` is a Boolean that specifies if the pseudo-volume is write-protected. `OLDDRIVERADDR` holds the address of this unit's (if assigned) previous driver address. It is used when normal floppy unit numbers are assigned to pseudo-volumes, so when released, the floppies can be reactivated. Each `REGULAR` entry is accessed via an index from 1 to 31. This index value is thus associated with a pseudo-volume. All references to pseudo-volumes in the Volume Manager are made with these indexes.

Immediately following the `VDIR` array is an array of description fields for each pseudo-volume:

```
        VDESC: ARRAY [0..31] OF STRING[15]
```

The description field is used to differentiate pseudo-volumes with the same name. It is set when the pseudo-volume is created. This array is accessed with the same index as `VDIR`.

The volume directory does not maintain the names of the pseudo-volumes. These are found in the directories in each pseudo-volume. When the Volume Manager is activated, it reads each pseudo-volume directory to construct an array of the pseudo-volume names:

```
        VNAMES: ARRAY [0..31] OF STRING[7]
```

Each pseudo-volume name is stored here so the Volume Manager can use it in its display of pseudo-volumes. The name is set when the pseudo-volume is created and can be changed by the Pascal Filer. The names in this array are accessed via the same index as `VDIR`. This array is set up when the Volume Manager is initialized and after there is a delete of a pseudo-volume. Creating a pseudo-volume will add to the array at the end.

Pascal Pseudo-Volume Format

Each Pascal pseudo-volume is a standard UCSD formatted volume. Blocks 0 and 1 are reserved for bootstrap loaders (which are irrelevant for pseudo-volumes). The directory for the volume is in blocks 2 through 5 of the pseudo-volume. When a pseudo-volume is created, the directory for that pseudo-volume is initialized with the following values:

```
dfirstblock = 0                first logical block of the volume
dlastblock = 6                 first available block after the directory
dvid  = name of the volume used in create
deovblk = size of volume specified in create
dnumfiles = 0                  no files yet
dloadtime = set to current system date
dlastboot = 0
```

The *Apple II Pascal 1.3 Manual* contains the format for the UCSD directory. Files within this subdirectory are allocated via the standard Pascal I/O routines in a contiguous manner.

Storage Type $5

Storage type $5 is used by the ProDOS FST in GS/OS to store extended files. The key block of the file points to an extended key block entry. The extended key block entry contains mini-directory entries for both the data fork and resource fork of the file. The mini-entry for the data fork is at offset +000 of the extended key block, and the mini-entry for the resource fork is at offset +$100 (+256 decimal).

The format for mini-entries is as follows:

storage_type	(+000)	**Byte**	The standard ProDOS storage type for this fork of the file. Note that for regular directory entries, the storage type is the high nibble of a byte that contains the length of the filename as the low nibble. In mini-entries, the high nibble is reserved and must be zero, and the storage type is contained in the low nibble.
key_block	(+001)	**Word**	The block number of the key block of this fork. This value and the value of storage_type combine to determine how to find the data in the file, as documented in the *ProDOS 8 Technical Reference Manual*.
blocks_used	(+003)	**Word**	The number of blocks used by this fork of the file.
EOF	(+005)	**3 Bytes**	Three-byte value (least significant byte stored first) representing the end-of-file value for this fork of the file.

Immediately following the mini-entry for the data fork may be up to two eighteen-byte entries, each with part of the HFS Finder information for this file. The first entry stores the first 16 bytes of the Finder information, and the second entry stores the second 16 bytes. The format is as follows:

entry_size	(+008)	**Byte**	Size of this entry; must be 18 ($12).
entry_type	(+009)	**Byte**	Type of this entry—1 for FInfo (first 16 bytes of Finder information), 2 for xFInfo (second 16 bytes).
FInfo	(+010)	**16 Bytes**	First sixteen bytes of Finder Info.
entry_size	(+026)	**Byte**	Size of this entry; must be 18 ($12).
entry_type	(+027)	**Byte**	Type of this entry—1 for FInfo (first 16 bytes of Finder information), 2 for xFInfo (second 16 bytes).
xFInfo	(+028)	**16 Bytes**	Second sixteen bytes of Finder Info.

Note: Although the ProDOS FST under GS/OS will only create both of the mini-entries, as described above, the ProDOS File System Manager (ProDOS FSM) for the Macintosh, which is part of the Apple IIe Card v2.0 software, may create only **one** of the entries, so you may find an `entry_type` of 2 at offset +009 in the block. If one of the entries is missing, it should be considered to be all zeroes.

All remaining bytes in the extended key block are reserved and **must** be zero.

Further Reference

- *Apple II Pascal ProFile Manager Manual*

- *GS/OS Reference*
- *ProDOS 8 Technical Reference Manual*