Apple II
Technical Notes

Developer Technical Support

GS/OS
#13: *GS/OS Reference* Update

Revised by: Matt Deatherage May 1992
Written by: Matt Deatherage & Dave Lyons November 1990

This Technical Note corrects and updates the Addison-Wesley *Apple IIGS GS/OS Reference*. Previous versions from APDA labeled Volume 1 or 2 are obsolete, and should no longer be used.
Changes since December 1991: Added new information about `resource_eof` and `resource_blocks` parameters.

Chapter 4, "Accessing GS/OS Files"

Page 72: The System File Level: How to Protect an Open File From the Application

The class 1 `SetLevel` and `GetLevel` calls have a special option that allows you to open a file at an "internal" file level, so that it cannot be closed by an application making a `Close` call with reference number zero at any application level.

`GetLevel` and `SetLevel` actually accept two parameters, not just the one parameter (`level`) documented in Chapter 7. The second parameter, `level_mode`, is a **Word** that controls the internal range of the file level.

Only two values for `level_mode` are supported. A value of $8000 is the same as if the parameter wasn't present at all—the level calls behave just as documented in *GS/OS Reference*. A value of $0000 sets a special "system" or "internal" level—all files opened with an internal level are unaffected by any non-internal level.

The steps to open a file at an internal file level are:

1. Call `GetLevel` with `pCount`=2, `level_mode`=$0000. Save the returned level.
2. Call `SetLevel` with `pCount`=2, `level` = $0080 and `level_mode` = $0000.
3. Open a file or files with a class 0 or 1 `Open` call, or with `OpenResourceFile` (`OpenResourceFile` on System Software 5.0.4 and earlier does not try to protect your resource files from being accidentally closed by a `Close`(0)).
4. Call `SetLevel` with `pCount`=2, `level_mode`=$0000, and `level` = saved level.

You can use two parameters in all your level calls and set the second `level_mode` parameter to $8000 instead of omitting it if it will make writing your program easier.

To close your protected file, simply do a `Close` with the reference number. There is no need to fiddle with the file level when closing by reference number.

NDAs should close all their files at or before `DeskShutDown` time.

Chapter 6, "Working with System Information"

Page 92: Using the `optionList` parameter

The `optionList` parameter resembles a GS/OS output buffer in most important respects—it starts with a word indicating the size of the buffer, and each FST fills in the size of the actual data placed in the buffer in the second word. If the buffer is too small to hold the data, the necessary size is placed in the second word and the FST returns the "buffer too small" error ($004F).

Usually, GS/OS input buffers only have one length word, because if you know how large the data is (and you do if you're the one passing it to GS/OS), you don't need another word telling you the same thing. However, if you're trying to **copy** something like an `optionList`, you can wind up in a bit of a pickle. Just because the buffer you've allocated is big enough to hold file system-specific information, that doesn't mean the information is necessarily present.

A good example of this problem is found in the System Software 6.0 ProDOS FST. In 6.0 and later, the ProDOS FST will take HFS Finder information (as returned by the AppleShare and HFS FSTs) in the `optionList` and place that information in an extended file's extended key block, so the file can be copied to and from ProDOS disks with no loss of Macintosh-specific information (such as the longer file types and creator types necessary to identify Macintosh files). The FST returns the same information (if present) in the output `optionList`.

However, previous versions of the ProDOS FST returned no information in the `optionList`. Suppose you archived a file and stored the `optionList` with the file's information under 5.0, and attempt to restore the file under 6.0 using a nice, large `optionList` buffer. The FST can't know whether the large buffer contains any information or not.

To remedy this problem, the second word of the `optionList` structure (`reqSize` in the figure on page 92) is now defined on **input** as well as output. On input, the word must contain the actual size of the data in the `optionList`; the first word continues to indicate the size of the entire buffer. If the buffer size and the actual data size are too small to make sense, any affected FSTs will ignore the input, knowing that it must be garbage.

Further details on how the ProDOS FST stores HFS Finder information can be found in ProDOS 8 Technical Note #25, "Non-Standard Storage Types."

Chapter 7, "GS/OS Call Reference"

Pages 98-99: ChangePath

On page 98, the *Reference* states that a subdirectory may not be moved into itself or into a directory the first subdirectory already contains. For example, you may not change `/v` to `/v/w` or `/v/w` to `/v/w/x`. Although this is correct, the System Software 5.0.x implementations of the ProDOS FST trash your disk if you try this with `ChangePath`. Do not try it on disks you want to keep.

On page 99, error $4E is described as "file not destroy-enabled." No, `ChangePath` doesn't destroy the file. The error should read "file not rename-enabled."

Page 120: DInfo Characteristics Word

The diagram for the `characteristics` word in the `DInfo` parameters has incorrect descriptions for bits 14 and 13. The diagram says bit 14 is set if the device is a linked device; in

fact, bit **13** is set if the device is a linked device. Bit 14 is set if the device in question has a generated driver; the bit is clear for loaded drivers.

Page 129: The Character Device Status Word

The diagram on the top of page 129 says that if bit 5 is set, the device is in no-wait mode. This is incorrect. To determine if a device is in no-wait mode, make the `GetWaitStatus` subcall described on page 130.

Bit 5 of the character device status word is set if there are one or more characters waiting to be read from the device. This is an assistance for developers, since generated character drivers don't support no-wait mode.

Page 132: GetFormatOptions Flags Word

The diagram describing the `flags` word of `GetFormatOptions` is incorrect. Bits 0 and 1 are actually the format type, while bits 2 and 3 are the size multiplier. In other words, the two labels are backwards.

Page 142: Flush

The `Flush` call, under System Software 5.0.3 and later (GS/OS version 3.3) accepts a maximum of two parameters. If the second parameter is present, it is the `flushType`. The `flushType` **Word** specifies the type of flush to be performed. A `flushType` of $0000 is the standard flush, where all dirty blocks are written to disk. If `flushType` is $8000, however, only dirty data blocks are written to disk. Certain dirty system blocks (blocks that don't hold file data) may not be flushed in this fast flush, but volume and file integrity is maintained.

Page 151: GetDirEntry
Page 156: GetFileInfo
Page 176: Open

Each of the above calls has optional `resourceEOF` and `resourceBlocks` paramters that are listed as "undefined" if the file has no resource fork. In System Software 6.0 and later, these fields are guaranteed to be zero if a given file has no resource fork.

Appendix A, "GS/OS ProDOS 16 Calls"

Page 386: GetDirEntry buffer description incorrect

On page 386, `nameBuffer` is described as a pointer to a buffer in which GS/OS returns a Pascal string containing the name of the file or directory entry (in `GetDirEntry`). This is incorrect; all versions of `GetDirEntry` return GS/OS (word-length) strings for the directory entry.

Further Reference

- *GS/OS Reference*
- Apple IIGS Technical Note #71, DA Tips and Techniques
- ProDOS 8 Technical Note #25, Non-Standard Storage Types