Apple II
Technical Notes

Developer Technical Support

GS/OS
#1: Contents of System Software Distribution Disks

Revised by: Matt Deatherage — June 1992
Written by: Matt Deatherage — November 1988

This Technical Note describes the contents of the disks System.Disk and System.Tools and the minimum files necessary to boot GS/OS starting with System Software 5.0.
Changes since January 1991: Now describes System Software 6.0. Changed the title to not reflect disk names.

This Note gives a description of each of the files in the Apple IIGS System Software 6.0 package. This package includes six disks: Install, SystemTools1, SystemTools2, Fonts, synthLAB and System.Disk. System Software 6.0 requires at least 1 MB of memory, one 3.5" drive and another storage device (either a second 3.5" drive or a larger capacity device). 2 MB of memory and a hard disk are highly recommended.

System.Disk is a pre-configured boot disk for floppy-based users. Because all the files on System.Disk appear on other disks in the 6.0 set, they are only listed and not described a second time.

Contents of Install

ProDOS	Every file system boots differently; the boot blocks for ProDOS disks look for a file name ProDOS. This is that file. It is the GS/OS file system stub necessary to start the boot process.
System	The directory containing most of the GS/OS files.
CDevs	The directory containing all Apple IIGS Control Panel Devices (CDevs) required for installing 6.0.
General	Allows setting of general system parameters.
RAM	Controls the size of the RAM disk and the GS/OS Disk Cache.
SetStart	Lets you choose which application to boot into.
Desk.Accs	The directory containing all the classic and new desk accessory files to be loaded at boot time.
ControlPanel	The New Desk Accessory which allows users to control almost all system parameters and choose printers and file servers.
Drivers	The directory containing all device drivers needed by GS/OS and the Toolbox (including the Print Manager and MIDI Tools).
AppleDisk3.5	The Apple 3.5 Drive device driver for GS/OS. Also drives SuperDrives connected to the Apple II SuperDrive interface card.

AppleDisk5.25	The driver for Apple 5.25" disk drives, including Disk II drives and Apple UniDisk 5.25 drives. This driver is required for GS/OS to recognize 5.25" disk drives. In 6.0, it is up to 300% faster than in earlier versions of system software.
Console.Driver	The text screen and keyboard device driver for GS/OS.
SCSI.Manager	The GS/OS SCSI Manager, the supervisory driver that arbitrates hardware-level usage of Apple's Apple II SCSI cards.
SCSIHD.Driver	The GS/OS driver for SCSI hard disks. This driver is required for GS/OS to recognize SCSI hard disks.
UniDisk3.5	The GS/OS driver for UniDisk 3.5 drives. This driver is required for proper operation of UniDisk 3.5 drives. Using the UniDisk with GS/OS without this driver eventually corrupts media.
Error.Msg	A compiled file containing all error messages required by GS/OS. This file is separate from the GS.OS file to provide easier support for localization.
Fonts	The directory containing all system fonts to be used.
FastFont	This makes Shaston 8 text drawing much faster.
FSTs	The directory containing the file system translators to be loaded at boot time.
Char.FST	The character device FST.
Pro.FST	The ProDOS FST.
GS.OS	The remainder of GS/OS.
GS.OS.Dev	The GS/OS Device Manager and associated core routines. Separate from GS.OS for speed reasons.
P8	The ProDOS 8 operating system.
SetStart.data	An invisible file created by the SetStart Control Panel, indicating which application the system should boot into. On this disk, this points to the Installer.
Start	The boot program. If this file exists, GS/OS always launches it upon booting. Under 6.0, this program usually reads the SetStart.data file and launches the indicated application.
Start.GS.OS	The file containing the `GLoader` and `GQuit` routines. It loads the files GS.OS and GS.OS.Dev, which contain the rest of the operating system.
System.Setup	The directory containing all the initialization files to be executed at boot time.
Resource.Mgr	The Resource Manager. This is an initialization file; the design of the Resource Manager requires it to be present even when an application has not specifically loaded it. The system does not boot if this file is not present.
Sys.Resources	A file containing system resources, available to the system software and to applications.
Tool.Setup	A required file that loads files which contain all the patches to tools in ROM for ROM levels 01 (TS2) and 03 (TS3). Tool.Setup would attempt to load TS1 if executed on a machine with ROM level 00, but GS/OS does not boot on such a machine, therefore, TS1 is not included. Tool.Setup also contains patches common to both ROM 1 and ROM 3.

TS2	Patches to ROM tools for ROM 1.
TS3	Patches to ROM tools for ROM 3.
Tools	The directory containing tool files for all tools not in ROM.
Tool014	Window Manager .
Tool015	Menu Manager.
Tool016	Control Manager.
Tool018	QuickDraw Auxiliary.
Tool019	Print Manager.
Tool020	LineEdit.
Tool021	Dialog Manager.
Tool022	Scrap Manager.
Tool023	Standard File.
Tool027	Font Manager.
Tool028	List Manager.
Tool034	TextEdit.
Icons	The directory containing all the Finder's old-style icon files as well as new Desktop database files and file type descriptors.
FType.Apple	The file type names used by the Finder (on all systems).
Installer	The Apple IIGS Installer program. This program makes use of scripts found in the Scripts directory on this disk to install parts of the system, as well as third-party applications, without the user needing to copy individual files.
Scripts	This directory contains all the scripts for the Installer. On launch, the Installer looks in its parent directory for the Scripts directory and the scripts it contains. It also reads `MessageCenter` message #1.
A2.RAMCard	Script to install the driver for the Apple II Memory Expansion Card (the slot-based, or "slinky" card).
Adv.Disk.Util	Script to install the Advanced Disk Utility program.
Apple.Bowl	Script to install the Apple Bowl game.
Apple.MIDI	Script to install the Apple MIDI Interface driver and tool set.
AppleDisk5.25	Script to install the 5.25" disk driver for GS/OS.
AppleShare	Script to install AppleShare.
AppleShare3.5	Script that creates an 800K or 1440K GS/OS startup disk which contains AppleShare.
Archiver	Script to install Archiver, the new GS/OS-based backup program.
Aristotle.Patch	Script to install a change to Aristotle for easier class transition.
ATImageWriter	Script to install the ImageWriter printer driver for the Print Manager, as well as the files necessary to work with AppleTalk.
ATImageWriterLQ	Script to install the ImageWriter LQ printer driver for the Print Manager, as well as the files necessary to work with AppleTalk.
Calculator	Script to install the Calculator new desk accessory.
Card6850.MIDI	Script to install the 6850-based MIDI Interface card driver.

CDROM	Script to install the High Sierra FST as well as the SCSI Manager and SCSI CD-ROM driver for GS/OS.
CloseView	Script to install the CloseView NDA, which makes the screen more legible to some visually-impaired users.
DCImageWriter	Script to install the ImageWriter printer driver for the Print Manager, as well as the files necessary to connect it to a serial port.
DCImageWriterLQ	Script to install the ImageWriter LQ printer driver for the Print Manager, as well as the files necessary to connect it to a serial port.
DOS3.3.FST	Script to install the read-only DOS 3.3 file system translator.
Easy.Access	Script to install the EasyAccess init, which provides sticky keys and keyboard mouse to ROM 1 users.
Epson	Script to install the Epson printer driver for the Print Manager, as well as the parallel card driver.
Fonts	Script to install the minimum suggested font set.
Fonts.Max	Script to install all fonts provided with System 6.0.
Fonts.Std	Script to install the standard font set.
HFS.FST	Script to install the Hierarchical File System (HFS, used on the Macintosh) file system translator.
Inst.Sys.Min	Script to install a minimal GS/OS system on an 800K volume. Note that this is **different** than 5.0.x's "Inst.Sys.Min" script, the 6.0 version of which is in the file named "AppleShare3.5".
Inst.SysF.NoFin	Script to install a minimal GS/OS system,without the Finder, on a given destination volume.
Instal.Sys.File	Script to install a complete System Software 6.0 configuration, including new features, on a given destination volume.
LaserWriter	Script to install the LaserWriter printer driver for the Print Manager, as well as the files necessary to work with AppleTalk.
Local.Net.Boot	Script to create a 3.5" floppy disk with minimal system software that boots into a server selection program (the network "Start" program from SystemTools2).
MediaControl	Script to install the Media Control toolset and all Media Control drivers supplied with System 6.0.
MediaCtrl.CDSC	Script to install the Media Control toolset and the drivers to work with the Apple CD SC drive.
MediaCtrl.P2000	Script to install the Media Control toolset and the drivers to work with the Pioneer 2000 series laserdisc players.
MediaCtrl.P4000	Script to install the Media Control toolset and the drivers to work with the Pioneer 4000 series laserdisc players.
Namer	Script to install the printer Namer Control Panel. Namer II (a ProDOS 8 application) is not included with System 6.0.
Pascal.FST	Script to install the read-only Apple II Pascal file system translator.
Quick.Logoff	Script to add a quick logoff feature to AppleShare.

SCSI.Hard.Disk	Script to install the SCSI Manager and SCSI hard disk driver for GS/OS.
SCSI.Scanner	Script to install the SCSI Manager and SCSI scanner driver for GS/OS.
SCSI.Tape	Script to install the SCSI Manager and SCSI tape driver for GS/OS.
Server.Sys.File	Script to install System Software 6.0 on an AppleShare File Server.
Sounds.All	Script to install all sounds provided with System Software 6.0 into the “System:Sounds” folder of the designated volume.
StyleWriter	Script to install the StyleWriter printer driver for the Print Manager, as well as the files necessary to connect it to a serial port.
Teach	Script to install the application Teach, which displays and edits Teach files, text files, AppleWorks files, MacWrite files and Installer scripts.
UniDisk3.5	Script to install the UniDisk 3.5 driver for GS/OS.
VideoKeyboard	Script to install the Video Keyboard new desk accessory, which allows users to type by using the pointing device instead of the keyboard.
VideoMix	Script to install the latest versions of the Apple II VideoMix software and tools.

Contents of SystemTools1

Icons	Additional icons for the Finder. This folder is currently empty.
System	A directory containing additional parts of the system software.
Finder	The Apple IIGS Finder, version 6.0.
CDevs	Directory with additional Control Panel Devices.
DirectConnect	Allows selection of direct-connected printers.
Keyboard	Sets keyboard parameters.
Modem	Controls modem port settings.
Monitor	Sets 40-column or 80-column mode, monochrome or color mode, and the color of text, text background, and borders.
Printer	Controls printer port settings.
Slots	Allows selection of slot settings and startup slot.
Sound	Sets user preference for sound pitch and volume. Also allows the user to assign digitized sounds to events that happen while using the computer.
Time	Sets the internal clock’s time and display format and optionally tracks Daylight Savings Time.
Desk.Accs	Directory with additional desk accessories.
CDRemote	An updated version of the CD Remote new desk accessory which ships with the AppleCD SC.
FindFile	A new desk accessory that finds files on volumes GS/OS can read.
Calculator	A calculator new desk accessory.
Drivers	Directory with additional device drivers for GS/OS and the Toolbox.

	A2.RAMCard	The GS/OS driver for slot-based memory expansion cards. This driver is not required to use these cards with GS/OS, but it does provide a substantial speed improvement.
	Apple.MIDI	The Apple MIDI Interface driver for the MIDI Tools.
	Card6850.MIDI	The driver for 6850-based MIDI interface cards for the MIDI Tools.
	Epson	The Epson® printer driver for the Print Manager.
	ImageWriter	The ImageWriter driver for the Print Manager.
	ImageWriter.LQ	The ImageWriter LQ driver for the Print Manager. Starting with System Software 5.0.3, this driver uses all the capabilities of the ImageWriter LQ.
	Modem	The modem port driver for the Print Manager.
	Parallel.Card	A driver for some parallel printer interface cards for the Print Manager. This driver works with the Apple Parallel Interface Card, as well as several other parallel interface cards.
	Printer The printer port driver	for the Print Manager.
	SCSI.Manager	The GS/OS SCSI Manager, the supervisory driver that arbitrates hardware-level usage of Apple's Apple II SCSI cards.
	SCSICD.Driver	The GS/OS driver for the AppleCD SC drive. This driver is required for GS/OS to recognize CD-ROM drives.
	SCSIScan.Driver	The GS/OS driver for the Apple Scanner or OneScanner. This driver is required for GS/OS to recognize Apple's scanners.
	SCSITape.Driver	The GS/OS driver for the Apple Tape Backup 40SC. This driver is required for GS/OS to recognize Apple's now-discontinued Tape Backup 40 SC.
	StyleWriter	The StyleWriter driver for the Print Manager.
Fonts		Directory with additional fonts
	Courier.09	9-point Courier font.
	Courier.10	10-point Courier font.
	Courier.12	12-point Courier font.
	Courier.14	14-point Courier font.
	Courier.18	18-point Courier font.
	Courier.20	20-point Courier font.
	Courier.24	24-point Courier font.
	Geneva.10	10-point Geneva font.
	Geneva.12	12-point Geneva font.
	Geneva.14	14-point Geneva font.
	Geneva.16	16-point Geneva font.
	Geneva.18	18-point Geneva font.
	Geneva.20	20-point Geneva font.
	Geneva.24	24-point Geneva font.
	Helvetica.9	9-point Helvetica font.
	Helvetica.10	10-point Helvetica font.
	Helvetica.12	12-point Helvetica font.
	Helvetica.14	14-point Helvetica font.
	Helvetica.18	18-point Helvetica font.
	Helvetica.20	20-point Helvetica font.
	Helvetica.24	24-point Helvetica font.
	Shaston.16	16-point Shaston font.
	Times.09	9-point Times font.

Times.10	10-point Times font.
Times.12	12-point Times font.
Times.14	14-point Times font.
Times.18	18-point Times font.
Times.20	20-point Times font.
Times.24	24-point Times font.
Venice.12	12-point Venice font.
Venice.14	14-point Venice font.
Venice.24	24-point Venice font.
FSTs	Directory with additional File System Translators.
DOS.3.3.FST	The DOS 3.3 FST, which allows GS/OS to access 5.25” disks formatted in DOS 3.3 format. This FST is read-only; it only performs read operations.
HS.FST	The High Sierra FST, which allows GS/OS to access CD-ROM discs formatted in the international standard High Sierra or ISO 9660 formats. This FST is read-only; it only performs read operations.
HFS.FST	The HFS FST, which allows GS/OS to read and write any disk in the Macintosh’s HFS format.
Pascal.FST	The Apple II Pascal FST, which allows GS/OS to access any disk formatted in Apple II Pascal format. This FST is read-only; it only performs read operations.
Tools	Directory with additional tools.
Tool025	Note Synthesizer.
Tool026	Note Sequencer.
Tool029	ACE Tools.
Tool032	MIDI Tools.
Adv.Disk.Util	The Advanced Disk Utility program which allows for partitioning of SCSI hard disks, as well as erasing, initializing, and zeroing volumes or partitions.
BASIC.System	The ProDOS 8 BASIC command interpreter.

Contents of SystemTools2

Icons	Additional icons for the Finder. This folder is currently empty.
AppleTalk	This directory contains additional AppleTalk files and utilities for AppleShare and AppleTalk.
Boot.Driver	A driver for AppleShare that GS/OS loads before the other drivers are loaded and which remains resident in memory after the boot process is finished. Installed on servers by the Installer script Server.Sys.File.
Display.0	An update to the Aristotle program installed by the “Aristotle.Patch” script.
QuickLogoff	An initialization file used to add a quick logoff feature to AppleShare.
Start	The AppleShare startup program which is installed instead of the standard Start program on AppleShare volumes. It allows the user to log on and then launches the server startup program for the user’s machine.

System	A directory containing additional parts of the system software.
CDevs	Directory with additional Control Panel Devices.
AppleShare	Allows users to choose and log onto AppleShare file servers.
FolderPriv	Allows users to set default folder privileges on AppleShare file server volumes.
MediaControl	Allows users to set up the Media Control tool set and the drivers they wish to use.
Namer	Allows users to rename AppleTalk-based ImageWriter, ImageWriter LQ and LaserWriter printers.
NetPrinter	Allows users to choose AppleTalk-based ImageWriter, ImageWriter LQ and LaserWriter printers.
Desk.Accs	Directory with additional desk accessories.
MediaControl	A new desk accessory that's like a "super" remote control for all devices the Media Control toolset can control.
VideoKeyboard	A new desk accessory that allows users to type with the pointing device instead of with the keyboard.
VideoMix	An updated version of the VideoMix new desk accessory which ships with the Apple II Video Overlay Card.
Drivers	Directory with additional device drivers for GS/OS and the Toolbox.
AppleTalk	The AppleTalk port driver for the Print Manager. It works with either serial port when configured for AppleTalk.
ATalk	The main AppleTalk GS/OS driver.
ATP1.ATROM	AppleTalk protocols to patch the IIGS ROM.
ATP2.ATRAM	AppleTalk protocols not in ROM.
IWEM	PostScript® program which allows a LaserWriter emulate an ImageWriter. A user can load it into the LaserWriter with the LaserWriter Control Panel, and it is automatically invoked when printing through the slot associated with AppleTalk.
LaserWriter	The LaserWriter driver for the Print Manager. This driver works with any LaserWriter with PostScript. It does not work with the LaserWriter IISC or Personal LaserWriter LS. This driver doesn't always print color patterns correctly to PostScript Level 2 printers, such as the LaserWriter IIf, LaserWriter IIg or Personal LaserWriter NTR.
Media.Control	Drivers for the Media Control toolset
AppleCDSC	Media Control driver for the Apple CD SC drive.
Pioneer2000	Media Control driver for the Pioneer 2000 series of laserdisc players.
Pioneer4000	Media Control driver for the Pioneer 4000 series of laserdisc players.
SCC.Manager	The GS/OS supervisory driver that arbitrates hardware-level usage of the serial communications controller in the Apple IIGS.

Fonts	Directory with additional fonts. Currently, this directory on this disk is empty.
FSTs	Directory with additional file system translators.
AppleShare.FST	The AppleShare FST which allows GS/OS to access AppleShare file servers.
Sounds	A folder with sounds provided for the new Sound Control Panel. The file names are fairly self-explanatory; the sounds are not described here.
Ahh	
Doorbell	
Droplet	
Eastern	
Frog	
PipeOrgan	
Quack	
SimpleBeep	
Sosumi	
Swish	
Trumpets	
Whoosh	
System.Setup	Directory with additional initialization files.
AppleIIVOC.INIT	An initialization file used by the Apple IIGS Video Overlay Card tool set.
ATInit	The AppleTalk initialization file.
ATResponder	The AppleTalk Responder, used for AppleTalk network management.
CloseView	A new desk accessory (installed by an init) that magnifies the screen to make it more visible to some users with visual impairments.
EasyAccess	An initialization file that brings Sticky Keys and Keyboard Mouse to ROM 1 users.
EasyMount	An initialization file that creates file server aliases in the Finder.
Tools	Directory with additional tools.
Tool033	VideoMix toolset (for the Video Overlay Card).
Tool038	Media Control toolset.
Archiver	A GS/OS based backup and restore program.
Teach	A simple editor that uses TextEdit to display and edit text files, Teach files, Installer scripts and AppleWorks and MacWrite documents.
Read.Me	Last-minute news and information about the System Software. Read with Teach.
Shortcuts	A Teach file with time-saving system tips and information.

Contents of Fonts

Goodies	A directory with files that are only related to system software in the vaguest sense.
Apple.Bowl	A GS/OS conversion of an old Apple II bowling game.
Read.Me	Documentation on Apple Bowl.
Icons	Additional icons for the Finder.
AppleBowl.Icon	The icon for the Apple Bowl game.
System	A directory containing additional parts of the system software.
Fonts	Additional fonts.
Courier.27	27-point Courier font.
Courier.28	28-point Courier font.
Courier.30	30-point Courier font.
Courier.36	36-point Courier font.
Courier.42	42-point Courier font.
Helvetica.27	27-point Helvetica font.
Helvetica.28	28-point Helvetica font.
Helvetica.30	30-point Helvetica font.
Helvetica.36	36-point Helvetica font.
Helvetica.42	42-point Helvetica font.
Helvetica.48	48-point Helvetica font.
Helvetica.60	60-point Helvetica font.
Helvetica.72	72-point Helvetica font.
Helvetica.96	96-point Helvetica font.
Times.27	27-point Times font.
Times.28	28-point Times font.
Times.30	30-point Times font.
Times.36	36-point Times font.
Times.42	42-point Times font.
Times.48	48-point Times font.
Times.60	60-point Times font.
Times.72	72-point Times font.
Times.96	96-point Times font.

Contents of synthLAB

synthLAB	The synthLAB application, a demonstration sequencer for the MIDI Synth toolset.
Tool035	MIDI Synth toolset.
MIDI	The MIDI Control Panel. Lets you choose a MIDI driver.
Seq.and.Instr	A directory containing demonstration sequences (files that end in ".seq"), wave forms (files that end in ".wav") and sound banks (files that end in ".bnk") for use with synthLAB and MIDI Synth. The files are only listed; their sound is not described here.
Synth.bnk	
Synth.seq	
Synth.wav	
Bee.seq	

Capri.seq	
Combo.bnk	
Combo.wav	
Demo.bnk	
Demo.wav	
Fugue.seq	
Midsummer.seq	
Orch.bnk	
Orch.wav	
Piano.bnk	
Piano.wav	
Rhythm.seq	
Sonata.seq	
Reference	A Teach document with the electronic manual for synthLAB.

Contents of System.Disk

Files are only listed here; they are described earlier in this Note where they first appeared.

ProDOS	
System	
Start.GS.OS	
GS.OS	
Error.Msg	
GS.OS.Dev	
FSTs	
Pro.FST	
Char.FST	
Drivers	
AppleDisk3.5	
AppleDisk5.25	
Console.Driver	
System.Setup	
Tool.Setup	
TS2	
TS3	
Resource.Mgr	
Sys.Resources	
Desk.Accs	
ControlPanel	
CDevs	
Printer	
Time	
Start	This is the Finder, not the SetStart program or the AppleShare program.
Tools	
Tool014	
Tool015	
Tool016	
Tool018	
Tool019	
Tool020	
Tool021	

-
  -
    - Tool022
    - Tool023
    - Tool025
    - Tool027
    - Tool028
    - Tool034
  - Fonts
  - P8
- Icons
  - Ftype.Apple
- BASIC.System

Minimum GS/OS System Disk Requirements

The following files are required for GS/OS to boot from a local disk. This list does not address files needed by the Finder or the IIGS Toolbox. Those files only required in certain circumstances are noted as such. Those files that may be excluded **only** when disk space or memory limitations make it **absolutely** necessary are marked with asterisks (*).

ProDOS			
System			
	Start.GS.OS		
	GS.OS		
	GS.OS.Dev		
	Error.Msg		
	FSTs		
		Pro.FST	
		*HS.FST	Required for High Sierra or ISO 9660 discs.
		Char.FST	
		*AppleShare.FST	Required to use AppleShare file servers
		*DOS3.3.FST	Required to use DOS 3.3 disks
		*Pascal.FST	Required to use Apple II Pascal disks
		*HFS.FST	Required to use HFS disks
	Drivers		
		*AppleDisk3.5	Required for Apple 3.5 Drives or SuperDrives.
		*AppleDisk5.25	Required for 5.25” drives.
		*UniDisk3.5	Required for UniDisk 3.5 drives.
		*SCSI.Manager	Required for SCSI devices.
		*SCSIHD.Driver	Required for SCSI hard disks.
		*SCSICD.Driver	Required for AppleCD SC drives.
		*SCSIScan.Driver	Required for Apple scanners.
		*SCSITape.Driver	Required for Apple Tape backup.
		Console.Driver	
		*ATalk	Required for AppleTalk (including AppleShare).
		*ATP1.ATROM	Required for AppleTalk (including AppleShare).
		*ATP2.ATRAM	Required for AppleTalk (including AppleShare).
		*SCC.Manager	Required for AppleTalk (including AppleShare).
	System.Setup		
		Tool.Setup	
		TS2	
		TS3	
		Resource.Mgr	
		Sys.Resources	

CDevs	
*AppleShare	Required for selecting AppleShare file servers.
*NetPrinter	Required for choosing printers.
*DirectConnect	Required for choosing printers.
*General	
*RAM	Should **always** be included if space allows. Provides the only way to set the size of the GS/OS Disk Cache.
Desk.Accs	Required for desk accessories; any desk accessories should be installed in this directory.
*ControlPanel	Required if you ship any Control Panels (CDevs).
*Start	Must be present for GS/OS to boot or some other file that GS/OS can boot into must be present in its place.
Tools	Required for any of the RAM-based tools; any RAM-based tools should be installed in this directory.
Fonts	Required for the Font Manager.
*FastFont	This makes Shaston 8 text drawing much faster and should be included unless absolutely impossible.
*P8	Required for ProDOS 8.
*BASIC.System	Required for AppleSoft BASIC.

Further Reference

- *GS/OS Reference*
- Apple IIGS Technical Note #100, VersionVille

Epson is a registered trademark of Seiko Epson Corporation.
PostScript is a registered trademark of Adobe Systems, Incorporated.

Apple II
Technical Notes

Developer Technical Support

GS/OS
#2: GS/OS and the 80-Column Firmware

Written by: Matt Deatherage November 1988

This Technical Note discusses the changes in handling the 80-column firmware between GS/OS and ProDOS 16.

For compatibility with the Apple IIe, the Apple IIGS does not treat slot 3 like it treats other slots. Instead of using a bit in the Slot Register ($C02D) to control the mapping of ROM in slot 3 between the built-in 80-column firmware and any peripheral card physically in slot 3, the soft switches `SETINTC3ROM` ($C00A) and `SETSLOTC3ROM` ($C00B) are used instead. On the Apple IIe, these soft switches (referred to by the single label `SLOTC3ROM`) respectively map the ROM at $C300 to the internal 80-column firmware (which works with the auxiliary-slot 80-column card in most IIe computers) or to a peripheral card in slot 3. Note that writing to `SETSLOTC3ROM` on a IIe or IIGS with no card in slot 3 results in floating bus addresses in the $C300 space.

ProDOS 8 will not allow an Apple IIe or later model computer to have a card other than an 80-column card in slot 3. ProDOS 8 needs the 80-column firmware on a 128K machine for use in the /RAM driver, and the enhanced Apple IIe has some of the interrupt firmware in the $C300 space. When ProDOS 8 is loaded in an Apple IIe or later, it writes to `SETSLOTC3ROM` and looks at five identification bytes. If all five of these bytes do not match, ProDOS 8 will write to `SETINTC3ROM` to use the internal firmware. If all five bytes match, the external slot 3 ROM is left mapped in.

ProDOS 16 fell victim to a bug in ProDOS 8 versions 1.2 through 1.6 which always switched in the internal 80-column firmware, regardless of the user's Control Panel setting. GS/OS does not have this bug; a card in slot 3 of a IIGS other than an 80-column card will not be mapped out by GS/OS.

Application programmers who require the 80-column firmware should be familiar of the following points:

- If your program contains a routine to insure that the 80-column firmware is indeed available, it could be buggy. Since ProDOS 16 always made the 80-column firmware available, your routine to check that condition may never have been executed.
- If your program <u>requires</u> the 80-column firmware and it is not available, your program should display a message on the screen informing the user that he must

set Slot 3 in the Control Panel to Built-in Text Display for your program to execute, then gracefully exit. Switching the $C300 ROM space, even with the user's permission, is not recommended. Slot 3 could contain an operating GS/OS device, perhaps even the one your program was launched from. Remember, it is possible to boot GS/OS from slot 3.

Do not try to be clever in a situation like this. For example, do not go looking at ID bytes in slot 3 to try to determine the type of device present so that you can switch it out if you identify it as a non-disk device. Slot 3 could contain an active device being operated by a loaded GS/OS driver.

Your program should not ask the user's permission to switch ROM space between ports and slots (or in this case, the internal firmware versus the external card). That is why there is a Control Panel. Simply display a message informing the user that he must set Slot 3 in the Control Panel to Built-in Text Display for your program to execute. You may offer to change the battery RAM parameter for the user and restart the system (using the `OSShutdown` call), but under **no** circumstances should you hit the soft switch yourself, even with the user's permission.

Further Reference

- *GS/OS Reference*, Volume 1
- ProDOS 8 Technical Note #15, How ProDOS 8 Treats Slot 3

GS/OS
#3: Pointers on Caching

Written by: Matt Deatherage

November 1988

This Technical Note discusses effective use of the GS/OS cache.

Introduction

GS/OS is the first Apple II operating system to offer a sophisticated caching mechanism. However, using the cache and using it **wisely** are two different things. This Note presents some concepts which should lead to higher performance for your application if it uses the cache.

What's Cached Automatically?

All blocks on a GS/OS readable disk could be classified into one of two categories. "Application blocks" are all blocks on the disk contained in any file (except a directory file), while "system blocks" are other blocks on the disk. System blocks belong to the file system and include directory blocks, bitmap blocks, and other housekeeping blocks specific to the file system.

GS/OS always maintains at least a 16K cache, even if the user has set the disk cache size to 0K with the Disk Cache new desk accessory. When the system (usually an FST) goes to read a system block, the block is identified as a candidate for caching and is cached if possible. Applications define blocks as candidates for caching by using the `cachePriority` field of many class 1 GS/OS calls. Note that class 0 calls do not have this field, thus applications using exclusively class 0 calls will not be able to cache any application blocks.

Although this difference may seem like a limitation, it in fact improves performance. On the Macintosh, most applications that work with files (like database managers) leave the file with which they are working open while they need it; the file is only closed when the window containing it is closed. Apple II programs historically are quite different—they usually read an entire file at the beginning, modify it in memory, and write it when the save function is selected. A moment's thought will show that if GS/OS arbitrarily cached most or all application blocks, system blocks that would be used again (such as directory blocks) will be kicked out to make room for them. We will see that this is probably a bad thing to do.

How to Cache Effectively

The first tendency of many programmers is to attempt to completely cache any given file, but this usually leads to a degradation in performance, not an improvement. In small caches such strategies can slow the system to a crawl, and large caches offer no significant improvement. Remember that until the cache memory is needed, it is available to the system. The cache size for GS/OS as set by the user is the maximum to be allotted, not the minimum.

Suppose you are attempting to cache a 40K file (80 512-byte blocks). If the cache is set to less than 40K, the entire cache will be written through, kicking out all system blocks currently cached. A cache of this size slows system performance for little gain, since the entire file could not be cached anyway. Even if the cache is large enough to hold the entire file, you are needlessly taking twice the amount of memory with the same file (by reading it into memory you have obtained from the Memory Manager and by asking GS/OS to keep a copy in the cache).

It is evident that the system makes the best use of the cache automatically, freeing your application from the duty of caching system blocks, but there are certain instances where caching application data can improve system performance.

An application which does not limit document size to available memory will often only keep a portion of the document in memory at any given time. Suppose that the beginning of such an application's document file contains a header which to various parts of the document file. (These parts could be chapters for a word processor, report formats for a database manager, or individual pictures for an animation program.) This document header is probably not very long, but the application will likely need to read it quite often to quickly access various portions of the document file.

This header is a prime candidate for caching since it is a part of the file which will definitely be read many times during the life of the application. Contrast this with arbitrarily caching the entire file, which needlessly wastes both cache space and available memory to keep a duplicate copy of something that may or may not be read from disk again.

Although caching provides enormous benefits to GS/OS, indiscriminate use of the cache will waste memory and degrade overall system performance. Be prudent and limit your use of the cache to those portions of your document files which will be read from disk many times.

Further Reference

- *GS/OS Reference*, Volume 1

GS/OS
#4: A GS/OS State of Mind

Revised by: Matt Deatherage March 1991
Written by: Matt Deatherage January 1989

This Technical Note discusses GS/OS concepts and practices.
Changes singe July 1989: Includes more information about thinking for non-ProDOS file systems.

Although GS/OS bears many similarities to ProDOS, GS/OS is a much wider-reaching operating system, working not only with multiple file systems but also with character devices. Some things which work under ProDOS cause problems under GS/OS, and application programmers need to be aware of the differences, particularly those developing text-based programs.

GS/OS Hints

Be aware of character devices. A legal GS/OS pathname, perhaps entered by a user in response to a prompt, could map to a character device, with potentially disastrous results. Error $58, `Not a Block Device`, can protect you against this on many calls, including `Create`, but you must still take precaution. `DInfo` tells you if a device is a character device or block device; bit seven of the `characteristics` word is set if the device is a block device.

Don’t preprocess pathnames. A user input routine which prevents users from entering pathnames that don’t follow ProDOS syntax may help prevent `Illegal Pathname Syntax` errors, but it also keeps users from creating files on non-ProDOS disks with anything but ProDOS pathname syntax, and it could keep them from accessing files on non-ProDOS disks which they created with another GS/OS application. Since the only FST which allowed you to write to a device under System Software 4.0 was ProDOS, you didn’t see this problem right away. However, System Software 5.0 includes an AppleShare FST which, compared to ProDOS, is fast and loose with pathnames. “How about an anti-ProDOS name?” is a legal AppleShare filename. To allow compatibility with present and future non-ProDOS FSTs, Apple suggests you pass user-entered pathnames directly to GS/OS, with no application preprocessing.

Remember that under GS/OS both colons and slashes are valid separators, and colons can only be separators. In addition, all eight bits of each byte of a pathname are significant. Refer to *GS/OS Reference*, Volume 1 for more information on GS/OS pathname syntax. Using all eight bits of each byte may be particularly difficult for text-based applications, which have no way to force the standard Apple II character set to display characters such as sigma (∑) or the copyright symbol (©); they can fiddle to get characters like the sterling pound sign (£) and an Apple (). Some programs may wish to adopt special typographical conventions for these special characters while others may choose not to create files with such characters in their names. These programs could present the user with a list of existing filenames (with some substitution for the characters which are unavailable), while providing a method of choosing one, to retrieve such files. Any way around this problem for a text-based program will be less than optimal.

Avoid the Text Tools and all slot dependencies. Preliminary GS/OS documentation points to a System Service call named `DYN_SLOT_ARBITER`. This mechanism, which is not fully implemented in System Software 5.0, eventually will allow the operating system to use internal ports and external slots for the same "slot" in the same session, instead of requiring the user to reboot the system to safely change between ports and slots. Applications which have hard-coded slot dependencies (as the Text Tools unfortunately require) make this transition very difficult, both for GS/OS and for the applications and users. We recommend that applications use the GS/OS loaded and generated character device drivers for text output. A `DInfo` call will tell you what slot or port a driver controls, and whether or not it is a character device.

Avoid other file system dependencies. Many of the things ProDOS programmers are used to as facts of life just are not true any longer. For example, filenames don't have to be 15 characters or less under GS/OS. When making class one calls, GS/OS will tell you if you don't have enough room for the pathname by returning a `Buffer Too Small` error ($4F). Avoiding file system dependencies means handling this error intelligently: if you receive it, allocate more space for the buffer and try the call again. GS/OS will tell you how much space is needed. If you absolutely must hard code pathnames, such as volume names, be sure to use the colon as the separator, because if you do not, filenames with slashes will cause problems. Similarly, don't assume any of the following:

- There can only be 51 files in the volume directory
- All devices are named "`.Dn,`" where `n` is the device number
- All blocks are 512 bytes long
- All devices are block devices
- Any other ProDOS-specific characteristics

Your application may have hidden file system assumptions as well. For example, while a directory behaves like a directory under all GS/OS file system translators, reading from a directory is not always as fast as it is for ProDOS disks. ProDOS directories are fairly linear and can be searched quickly; but other file systems may have more complicated directory structures (HFS and AppleShare, for example, have B-trees that store directory entries in alphabetical order). To get optimal speed, try to do as many `GetDirEntry` calls as you can in succession without other GS/OS calls intervening—this allows Apple to optimize file system translators for fast directory reading.

Also remember that other file systems may not support the concept of orderable directories, so don't depend on directory order in your application.

Don't hog all of the memory. While this is never a good idea on the IIGS, it's even worse under GS/OS. To process things like pathnames, GS/OS allocates memory through the Memory

Manager. If you've allocated all of available memory (i.e., for a disk copy procedure), GS/OS will be forced to return an `Out of Memory` error ($54). If the condition is so severe that GS/OS can no longer function, it will return a fatal GS/OS error with an ID = 2, and the user will be asked to restart the system.

(A common cause of fatal GS/OS error 2 during development is using a length byte instead of a length word on a class one string. Doing so almost always causes the first word to be greater than 8K, which is the maximum length of pathnames under GS/OS. GS/OS then dies for your enjoyment, as it is unable to allocate the memory for the pathname because it's too big, even if more than 8K is available.)

Hard code as little as possible. Even seemingly static things like device names should not be hard coded, since a new loaded driver could change the name of the same device at any time. Also, it may be possible in the future for users to rename devices.

Only ask for the access you need. If you're just going to read a file, make a call to `Open` the file with read permission only. In file systems where access privileges mean more than they traditionally have in ProDOS (where things are usually "Locked" or "Unlocked"), this could save some trouble. For example, AppleShare allows the same file to be opened multiple times as long as each open is with read-only access. If your program is only going to read a file, opening it with read and write access needlessly denies others on the server access to the file.

Copy all GS/OS information with files. Applications that copy files need to do more than copy the data fork of the file. If the file is extended, the resource fork of the file should be copied as well. In addition, when requested, each FST returns an `option_list` that contains information specific to the host file system that GS/OS does not use (i.e., AppleShare's `option_list` includes Finder information and access privileges). Calls to `GetFileInfo` and `Open` can return the `option_list`, while a call to `SetFileInfo` can set it. An FST will not set parameters in the `option_list` which should not be altered (just as `SetFileInfo` skips the EOF fields in `GetFileInfo` records). To ensure that the duplicate has as much host file system information from the original as can reasonably be transferred, always copy the `option_list`.

However, if you **want** to change something in an existing file's `GetFileInfo` list, do **not** use an `option_list`. The `option_list` could override the other parameters to `SetFileInfo` without your knowledge.

Further Reference

- *GS/OS Reference*, Volumes 1 and 2

GS/OS
#5: Resource Fork Formats

Revised by: Matt Deatherage July 1989
Written by: Matt Deatherage January 1989

This Technical Note discusses the resource fork format of GS/OS extended files.
Changes since January 1989: Documented the location of resource fork format information.

Due to an omission in *GS/OS Reference*, Volume 1, some developers are not aware that the format of the resource fork of **any** file is reserved by Apple Computer, Inc. With the release of System Software 5.0 for the Apple IIGS, a Resource Manager is available to manipulate discrete chunks of data stored in the resource forks of files. To prevent corruption of media, information should **only** be stored in any resource fork in this format.

The Resource Manager should always be used to manipulate the data in resource forks. Some utilities may find this impossible and will require direct manipulation of resources without the Resource Manager. Information on the format of the resource forks is in the Resource Manager chapter of Volume 3 of the *Apple IIGS Toolbox Reference*.

Further Reference

- *GS/OS Reference*
- *Apple IIGS Toolbox Reference*, Volume 3

GS/OS
#6: Drivers and GS/OS Direct Page

Revised by: Matt Deatherage & Dave Lyons November 1990
Written by: Matt Deatherage March 1989

This Technical Note corrects an error in the preliminary GS/OS documentation and provides an alternate suggestion for developers who are writing GS/OS drivers.
Changes since July 1989: Updated the list of calls which do not require the GS/OS direct page and updated the documentation references.

Preliminary GS/OS documentation, including the beta draft of *GS/OS Reference*, Volume 2, incorrectly states that locations $5A through $5F are available for device drivers, and that locations $66 through $6B are shared by device drivers and supervisory drivers (and may be corrupted by either a driver or supervisory driver call).

This is **not** correct. The locations in question are used by GS/OS; destroying these locations can cause system failure and media corruption.

Drivers which require direct page space of their own should request it from the Memory Manager when they are started. Upon receiving a call, a driver can save the value of the `D` register (containing the GS/OS direct page) and switch to its own direct page. The driver may keep the value of its direct page inside the driver itself; no space on GS/OS direct page is available for this purpose. The driver must restore the `D` register to point to the GS/OS direct page before returning from the call, and it should also dispose of its direct page space when it shuts down.

The driver must also set the `D` register to point to the GS/OS direct page before making any system service call **other** than `SET_SPEED`, `DYN_SLOT_ARBITER`, `MOVE_INFO`, `SIGNAL`, and `INSTALL_DRIVER`.

Note: The location of the GS/OS direct page is guaranteed to remain the same between `Driver_StartUp` and `Driver_ShutDown` calls.

Further Reference

- *GS/OS Device Driver Reference*

Apple II
Technical Notes

Developer Technical Support

GS/OS
#7: Behavior of SET_DISKSW

Written by: Matt Deatherage July 1989

This Technical Note discusses changes to the documented behavior of `SET_DISKSW` in System Software 5.0. This Note is primarily of interest to device driver authors.

GS/OS Reference, Volume 2, states that the system service call `SET_DISKSW` ($01FC90) will remove a device’s blocks from the cache and place its volumes off line.

With System Software 5.0, this behavior is slightly changed. `SET_DISKSW` also posts insertion and ejection notices to the GS/OS Notify Procedure queue, so that notification procedures may be called. This requires `SET_DISKSW` to check the current status of the device to know if the disk switched condition indicates an insertion or an ejection (by comparing the current device status against the device-dispatcher maintained status).

A GS/OS driver may have an interrupt handler present to handle interrupts generated by its device on insertion or ejection (if the hardware is capable of generating such interrupts). Such an interrupt handler will probably want to call `SET_DISKSW` when an insertion or ejection is detected to make the rest of the operating system aware of it. However, `SET_DISKSW` obtains the device’s status based on the `deviceNum` and `callNum` on the GS/OS direct page.

Any driver or interrupt handler calling `SET_DISKSW` must first save the values for `deviceNum` and `callNum` on the GS/OS direct page, replacing `callNum` with the number of a driver call that accesses media (Apple suggests `Driver_Read`, $0002) and replacing `deviceNum` with the number of the device for which `SET_DISKSW` is being called. The caller must restore the original values after `SET_DISKSW` returns.

Although `SET_DISKSW` saves and restores the GS/OS direct page, the caller must know where the GS/OS direct page is located so it can place the proper parameters there. The value used for the GS/OS direct page should be the value of the `D` register when the driver receives its `Driver_StartUp` call. The GS/OS direct page is now guaranteed to remain constant between `Driver_StartUp` and `Driver_ShutDown` calls.

Further Reference

- *GS/OS Reference*, Volume 2

Apple II
Technical Notes

Developer Technical Support

GS/OS
#8: Filenames With More Than CAPS and Numerals

Written by: Matt Deatherage July 1989

This Technical Note discusses the problems some applications may have when dealing with filenames containing lowercase letters for the first time.

With System Software 5.0, lowercase filenames enter GS/OS *en masse* for the first time. Lowercase filenames are inherent to the AppleShare filing system and have been added to the ProDOS filing system through the ProDOS FST. However, since Apple II filing systems never had lowercase characters in filenames before, this change undoubtedly causes problems for some applications. This Note gives general guidelines to help developers avoid such problems.

How the ProDOS FST Does It

“Wait,” you say (not for any particular reason, other than a general fondness for monosyllables). “If you put lowercase characters in the ProDOS directory entry, it’s going to cause all kinds of problems. What’s gonna’ happen on][+ machines?”

Two previously unused bytes in each file’s directory entry are now used to indicate the case of a filename. The bytes are at relative locations +$1C and +$1D in each directory entry, and were previously labeled `version` and `min_version`. Since ProDOS 8 never actually used these bytes for version checking (except in one case, discussed below), they are now used to store lowercase information. (In the Volume header, bytes +$1A and +$1B are used instead.)

If `version` is read as a word value, bit 7 of `min_version` would be the highest bit (bit 15) of the word. If that bit is set, the remaining 15 bits of the word are interpreted as flags that indicate whether the corresponding character in the filename is uppercase or lowercase, with set indicating lowercase. For example, the filename `Desk.Accs` has a value in this word of $B9C0, or binary 1011 1001 1100 0000. The following illustration shows the relationship between the bits and the filename:

```
Bits in WORD:              1011100111000000
Filename:                   Desk.Accs
Uppercase or Lowercase:     ULLLUULLL
```

Note that the period (.) is considered an uppercase character.

What it Means

Because no lowercase ASCII characters are actually stored in the filename fields of the directory entries, all ProDOS 8 software should continue to work correctly with disks containing files with lowercase characters in the filenames. Neither ProDOS 8 nor the ProDOS FST are case sensitive when searching for filenames: ProDOS is the same file as PRODOS is the same file as prodos.

The main trouble applications have is when a filename has been "processed" by the application before passing it to GS/OS. For example, if a command shell automatically converts filenames to all uppercase characters before passing them to ProDOS 16, the chosen uppercase and lowercase combination for the filename will never be seen by the user without any apparent reason. Some developers have considered it okay to ignore lowercase considerations, thinking that they would only apply to file systems other than ProDOS (and file systems which would not be available on the Apple II for a long time, if ever). These developers were mistaken.

A more pressing problem is that of an application that is looking for a specific file, perhaps a data file or a configuration file. If the application simply passes a pathname to GS/OS and asks for that file to be opened, it will be opened if it exists. The case of the filename is irrelevant since file systems are not case sensitive. However, if the application makes `GetDirEntry` calls on a specific directory, looking for the filename in question, there could be trouble: the application won't find the file unless its string comparison routine is not case sensitive. If the user has renamed the file MyApp.Config, and the string comparison is looking for MYAPP.CONFIG, then the application will report that the file does not exist.

It is repeated here that when dealing with normal OS considerations, it's almost always better to ask for something and respond intelligently if it's not there than it is to go looking for it yourself. The OS already has a lot of code to look for things (or expand pathnames, or examine access privileges, etc.), and reinventing the wheel is not only tedious, it can be detrimental to future compatibility.

The One Exception

In the past, ProDOS 8 did look at the version bytes when opening a subdirectory. The code to do this has been removed from ProDOS 8 V1.8. Please be aware that earlier versions of ProDOS 8 will be unable to scan subdirectories with lowercase characters in the directory name, even to find files in those directories.

Conclusion

Most user-input routines (including the Standard File tool set) return filenames or pathnames that can be passed directly to GS/OS without preprocessing. Doing so may return "pathname syntax errors" more often than not doing so, but it also enables applications to take advantage of future versions of the System Software that loosen the restrictions on syntax (or new file systems that never had such restrictions). Under GS/OS, even ProDOS disks aren't what they used to be.

Further Reference

- *GS/OS Reference*

Apple II Technical Notes

Developer Technical Support

GS/OS
#9: Interrupt Handling Anomalies

Revised by: Matt Deatherage — May 1992
Written by: Dave Lyons — January 1990

This Technical Note discusses anomalies in the way GS/OS handles interrupts.
Changes since May 1990: Added discussions about changes to GS/OS interrupt handling since System Software 5.0.2.

Problems Installing Interrupt Handlers

If your application calls `ALLOC_INT` to install an interrupt handler for an external interrupt source, it works fine **unless** the SCSI Manager (GS/OS file SCSI.Manager) is installed, in which case the system eventually grinds to a halt with a message about 65536 unclaimed interrupts.

The Problems

If any interrupt handlers are bound (using `BindInt`) to reference number $17 (IRQ.OTHER), the unclaimed interrupt count gets incremented if none of the `BindInt` routines claims the interrupt, even though any handlers installed with `ALLOC_INT` routines still need a chance to claim it. The 5.0.2 SCSI.Manager triggers this problem because it calls `BindInt` with vector reference number $17.

In addition, if one or more interrupt handlers are bound to the IRQ.OTHER vector (VRN $17), the interrupt is passed to the `ALLOC_INT` handler even if it was already claimed by a `BindInt` routine. If no `ALLOC_INT` routine claims the interrupt, the unclaimed-interrupt count is incremented. As documented in Apple IIGS Technical Note #18, Do-It-Yourself SCC Interrupts, you cannot successfully call `BindInt` with vector reference number $0009.

The Solution

An application may install **both** a `BindInt` routine and an `ALLOC_INT` routine. If they both claim the external interrupt, the unclaimed count does not get incremented. The solution is compatible with future System Software releases, since it does not depend upon the `ALLOC_INT` routine ever getting called.

Your application's `BindInt` routine sees the interrupt before your `ALLOC_INT` routine does, so the `BindInt` routine should figure out whether the interrupt was caused by your external device, and claim it if so. Your `ALLOC_INT` routine should claim an interrupt it sees if and **only** if your `BindInt` routine claimed the last interrupt it saw.

Starting with GS/OS version 3.2 (released with the Apple II High-Speed SCSI Card), the system no longer treats too many unclaimed interrupts as a fatal error. However, before version 6.0, it still counts the unclaimed interrupts so it can do something like display a dialog asking you to restart even though choosing "restart" returns you to the application unharmed (GS/OS version 3.2), or sometimes display a dialog box sending you to your dealer and sometimes not (version 3.3), or do nothing about it at all (version 4.0 and later). This is obviously as confusing to most of us as it was to the system itself, so fortunately GS/OS now ignores unclaimed interrupts and doesn't even bother counting them.

Problems Removing Interrupts Handlers

The *GS/OS Reference* suite says that device drivers may make `BindInt` and `UnbindInt` calls, noting this as an exception to the general rule that drivers may not make GS/OS system calls. What the references fail to note is that these calls may fail for an incredibly annoying reason—the OS may be busy.

GS/OS takes special pains to avoid this while starting and while switching to ProDOS 8, but it does not avoid this condition during an `OSShutDown`—a real shutdown of the OS, not a switch to ProDOS 8.

Driver authors can work around this problem by using a new system service call provided in GS/OS version 3.2 and later. The call, named `UNBIND_INT_VECTOR`, provides the functionality of `UnbindInt` to FSTs and drivers **only** to avoid the OS reentrancy issue. The vector is at $01/FCD8 and takes an interrupt identification number (as returned from `BindInt`) in the accumulator.

Further Reference

- *GS/OS Reference*
- Apple IIGS Technical Note #18, Do-It-Yourself SCC Interrupts

Developer Technical Support

GS/OS
#10: How Applications Find Their Files

Revised by: Matt Deatherage May 1992
Written by: Dave Lyons January 1990

This Technical Note explains how applications should find configuration and other application-related files.
Changes since September 1990: Lists new ways to access the @ prefix under System Software 6.0 and later.

When an application is launched, GS/OS sets prefix 9 to the application's parent directory. It also sets prefix 1 to the same directory if the length of the pathname is within a 64-character limit. It does not set prefix 0 to any special value.

If your application uses a partial pathname and depends upon prefix 0 to find files at the same directory level, it may be working by accident (prefix 0 is accidently set to the right directory), and sooner or later it won't work.

If your application needs to load a file named TitleScreen, the best way is to use the pathname 9:TitleScreen. If you just use TitleScreen, you are using prefix 0, and you may or may not be looking in the right directory.

Files storing user-specific data should be stored in the at sign (@) prefix—this is just like prefix 9, except that it is set to the user's user folder on an AppleShare server if the application was launched from a server. Use @:MySettings rather than 9:MySettings or MySettings. (If you want to retrieve the value of the @ prefix, you can call `ExpandPath` on the pathname "@:".) Note that the @ prefix was introduced in System Software 5.0.

The @ prefix is useful only for applications, not for Desk Accessories, CDevs, initialization files, or anything else; this type of code can get the path of the user's folder by using the AppleShare FST's FST-Specific call `GetUserPath`.

Starting with System Software 6.0, you can also retrieve the value of the @ prefix by passing $FFFF (–1) to `GetPrefix`. You may also set the value of the @ prefix by passing $FFFF to `SetPrefix`, but only applications or system-wide utilities should **ever** change the @ prefix. Specifically, any DAs, CDevs, initialization files or others should not mess with the @ prefix to make their own file handling simpler.

Further Reference

- *GS/OS Reference*
- AppleTalk Technical Note #8, Using the @ Prefix

Apple II
Technical Notes

Developer Technical Support

GS/OS
#11: About EraseDisk and Format

Revised by: Matt Deatherage November 1990
Written by: Dave Lyons & Matt Deatherage July 1990

This Technical Note explains how an application can tell when a user chooses Cancel from an `EraseDisk` or `Format` dialog box and explains why the `file_sys_ID` field is ignored in class-zero calls.
Changes since July 1990: Noted that System Software 5.0.3 fixes some of these anomalies.

Detecting a Canceled Erase or Format Dialog Box

GS/OS Reference says that `EraseDisk` and `Format` return with the carry flag set and `A` equal to zero when the user cancels the operation. This is great, except that the calls actually return with the carry clear, making a Cancel hard to distinguish from a successful `EraseDisk` or `Format` operation. This happens in System Software 5.0.2 and earlier; it works as documented in *GS/OS Reference* in System Software 5.0.3 and later.

If you must use 5.0.2 or earlier versions of the system software, this Note presents a safe way around the problem, which works with all versions of the System Software:

1. In the parameter block for class-one `EraseDisk` or `Format`, set the `fileSysID` field to zero. (See note below.)
2. Make the call.
3. If the error code is non-zero, there was an error. Handle it.
4. Otherwise, the error code is zero. Check the `fileSysID` field in the parameter block. If it is still zero, the user chose to cancel the operation.

Note that this method only works for class-one calls. For the class-zero `ERASE_DISK` and `FORMAT` calls, the `file_sys_ID` word is only an input parameter and always remains unchanged.

About the Class-Zero file_sys_ID Parameter

Even though `fileSysID` is an **input** parameter for the class-zero calls `ERASE_DISK` and `FORMAT`, all versions of the system software ignore the supplied value and always give the user a dialog for selecting a file system. This means no functionality is lost by putting a zero there.

The reasons for this decision are historical. Although the *Apple IIGS ProDOS 16 Reference* indicates that the input parameter `file_sys_ID` would be used in future versions to choose destination file systems, ProDOS 16 always returned an error if the file system specified was not $0001 (ProDOS).

Since this effectively means no `ERASE_DISK` or `FORMAT` call can be made under ProDOS 16 with any `file_Sys_ID` other than $0001, the GS/OS team chose to ignore the parameter and always give users the choice when using class zero calls. Otherwise, no program that existed when GS/OS was released would ever allow users to choose interleaves or file systems (they would always format for ProDOS, file system $0001). (Note that the class-one `Format` and `EraseDisk` calls have a new `reqFileSysID` parameter; if this field is present, the dialog box is bypassed.)

Further Reference

- *GS/OS Reference*
- *Apple IIGS ProDOS 16 Reference*

Apple II
Technical Notes

Developer Technical Support

GS/OS
#12: All About Notify Procs

Written by: Matt Deatherage September 1990

This Technical Note discusses the GS/OS notification procedure new to System Software 5.0 and enhances the discussion of these procedures in the Addison-Wesley *GS/OS Reference*.

Why Do I Want To Be Notified?

GS/OS notification procedures (or "notify procs") are handy ways to let the operating system tell you when interesting things are happening. As documented in *GS/OS Reference*, they can tell you when you're switching to ProDOS 8 (and back), when disks are inserted or ejected, when GS/OS is shut down, and even when a change occurs to a volume.

However, getting these notifications is not as simple as installing a procedure. Some behaviors are due to the way device drivers are designed and some are due to the design of GS/OS or device hardware. This Note discusses a few slightly unusual situations you can encounter when dealing with notification procedures.

I Get "Parameter out of range," and There's Only One Parameter

It seems incongruous to get error $0053 ("Parameter out of range") when there's only one parameter, a pointer to the notification procedure. However, GS/OS checks the procedure header to ensure consistency. In particular, the `flags` field must not have any of the reserved bits set. Having any bits other than one through six set results in error $53; it ensures you do not get strange behavior or are not passed values you cannot comprehend.

I'm Not Getting Notified

You've written your notification procedure correctly and tested it, but when you run your application you can eject and insert disks until your arm falls off and your code is never called.

This is a side effect of the design of most Apple II peripherals—no hardware interrupt is generated when you eject a disk. Without an interrupt to grab the CPU's attention, the drive just sits there until someone actually asks the drive if a disk is present.

Well-designed GS/OS drivers look to see if a disk has been switched every time they get control and call the System Service routine `SET_DISKSW`, which in turn causes the notification procedures to be told the disk has been switched. However, the driver cannot set this chain in motion until it gets control.

The easiest way to do this is to loop through all on-line devices, issuing a device call to each in turn. When the driver gets control, it starts the ball rolling. Note that you must make a device call that actually causes driver code to be executed. This includes all the application level device calls with less than two parameters, except `DRename` and `DInfo` (the third parameter is a block count, which causes a `Driver_Status` call to the driver). These calls are handled entirely by the Device Manager without actually transferring control to any driver code. `DStatus` with a `transferCount = 2` is a good choice.

I Get Notified About Insertion at Weird Times

When coming back to GS/OS from ProDOS 8, you get “insertion” notification even though no disks have actually been inserted. This is done for you by most drivers, which pretend that any media in the device has just come online at driver startup time—which is true as far as any application is concerned.

General Truths

Be careful when installing notification procedures from an application. Applications either go away or are made purgeable when they quit, and that means your notification procedure can get disposed. GS/OS tries to call the address anyway, and this is generally a bad idea. Make sure you remove all notification procedures before their code goes away.

Even though you have to poll to ensure you get disk insertion and ejection events, it’s still useful to install notification procedures. The notification queue allows **everyone** who’s interested in GS/OS events to be notified about them. Check the “disk has been switched” bit of the status word is not suitable, because this bit is only set once. If a desk accessory makes a status call to a switched device, it sees the “disk has been switched” bit and your application does not, so use the notification queue.

Operating system calls (i.e., `Write`) can generate volume changed events during execution; therefore, GS/OS could be busy when it calls your notification procedure. Volume changed events are not necessarily generated immediately. The AppleShare FST checks for volume changes approximately every 10 seconds, but it only generates these events for a given volume if it contains an open folder.

GS/OS can call your notification procedure from inside an interrupt, so make it short and sweet. One approach is setting a flag which you can check periodically from your main code; when the flag is set, you can process the event and clear the flag.

Further Reference

- *GS/OS Reference*

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

GS/OS
#14: The Console Driver Technical Note

Written by: Matt Deatherage May 1992

This Technical Note discusses the GS/OS Console Driver and related issues.

New 6.0 Character Features Don't Work In Version 3.2

The System Software 6.0 documentation (as of this writing, the GS/OS ERS) refers to a new Console Driver feature. The Console Driver now has the capability to return direct character-in and character-out vectors for improved throughput (gained by bypassing most of GS/OS's overhead). The vectors are obtained through new `DStatus` device-specific call $8007, `GetVectors`.

Unfortunately, in version 3.2 of the Console Driver (which ships with System Software 6.0), this call returns addresses which are almost the correct ones (in other words, they're wrong). If `DInfo` says the Console Driver is version 3.2 or earlier, don't try to use the `GetVectors` feature.

No-Wait Mode and User Input Mode Conflict

When you read from a GS/OS driver in no-wait mode, the driver is supposed to return as quickly as possible, reading as much information as possible and returning as soon as the request is filled or no more information is instantly available. This is the opposite of wait mode, where the driver waits until the read can be finished even if it takes forever.

This philosophy directly conflicts with the Console Driver's user input routine (UIR) mode, where standard human interface editing functions are available. For example, if you want to read seven characters from the Console Driver in UIR mode, the user should be able to type four characters and hit three backspaces and not worry that the read request will end since he pressed seven keys. The entire concept of UIR mode is that the user can take his time and edit his input until he's happy with it, then press a terminator key to end editing.

This is how the Console Driver works, in fact, even in no-wait mode. If you ask for even one character in UIR mode and no-wait mode, the Console Driver will let the user edit the one character until he presses a terminator.

If you want instant feedback, you must use raw input mode.

Further Reference

- *GS/OS Reference*
- System 6.0 Documentation for GS/OS