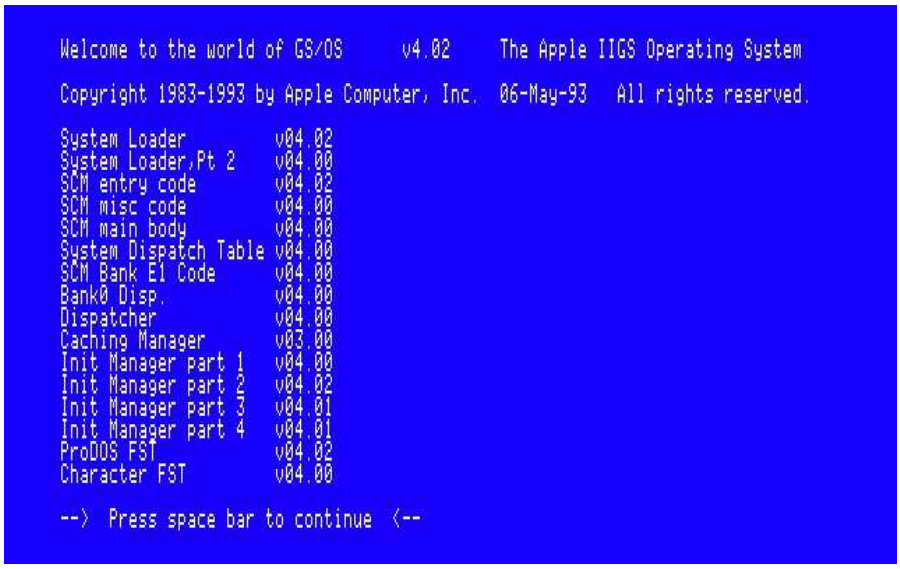


Apple IIgs®
GS/OS® Internals

Contents

This book is dedicated to the memory of Joe Kohn (1947-2010)
Joe was the reseller of Convert 3200, our graphic converter software

Preface

About this book

This book is for advanced Apple IIgs programmers and curious people. It is an unofficial publication from Brutal Deluxe Software and all the information given herein are related to the Apple IIgs System 6.0.1 version. This book is the result of a 3-years understanding of the inners of GS/OS which was done in order to build a File System Translator for the system.

This book must be seen as the GS/OS reference volume 2. It contains information described in the first volume, gathers data from external reference specifications and technotes.

Copyright information

Important note

The values and offsets in this book are given in hexadecimal format. A value of ’12’ must be understood as a decimal value of 18.

References

GS/OS reference volume 1 APDA-56
The official guide to developing software using the GS/OS operating system.

GS/OS device driver reference APDA-27
This reference describes the GS/OS application interface to device drivers.

Online resources

A2-Central http://www.a2central.com/
Your total source for Apple II computing.

Apple Archives http://www.applearchives.com/
The most complete and current Apple II related directory on the internet.

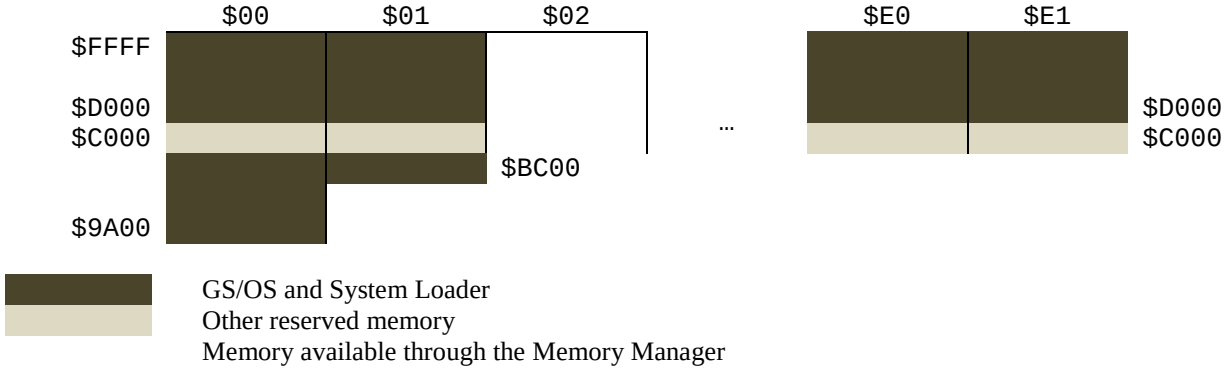
Brutal Deluxe Software http://www.brutal-deluxe.fr/
Website of a French group of IIgs programmers.

Syndicomm http://www.syndicomm.com/
The online resource to purchase the official Apple II documentation and software.

Memory usage

Apple IIGS memory map

The figure describes the memory map of the Apple IIgs that is used by GS/OS. The X-axis represents the bank numbers and the Y-axis represents the addresses within a bank.



GS/OS and System Loader
Other reserved memory
Memory available through the Memory Manager

GS/OS memory map

Module name	Version	Memory location (length)
System Loader	4.02	$01/A600 ($1525)
System Loader, Pt 2	4.00	$01/D000 ($2B49)
SCM entry code	4.02	$00/9A00 ($087B)
SCM misc code	4.00	$00/B300 ($06EC)
SCM main body	4.00	$00/D000 ($2FFA)
System Dispatch Table	4.00	$01/FC00 ($0100)
SCM Bank E1 Code	4.00	$E1/D980 ($067E)
Bank0 Disp.	4.00	$00/A000 ($07E0)
Dispatcher	4.00	$E0/E000 ($1F39)
Caching Manager	3.00	$00/A280 ($0764)
Init Manager part 1	4.00	$00/B200 ($00E4)
Init Manager part 2	4.02	$00/D400* ($0B6F)
Init Manager part 3	4.01	$01/D000* ($0FC3)
Init Manager part 4	4.01	$E0/D400* ($03E1)

* alternate bank

GS/OS vector space

Name	Address	Length	Description
OS_Entry	$E1/00A8	4	GSOS Entry vector for in-line GS/OS system calls
OS_Switch	$E1/00AC	4	Reserved for internal use
OS_StackEntry	$E1/00B0	4	GSOS2 Entry vector for stack-based GS/OS system calls
OS_Internal	$E1/00B4	4	OS_P8_SWITCH

GS/OS bank E1 globals

Name	Address	Length	Description
OS_Public_Flags	$E1/00B8	1	Bit 15: 0 = Standard boot 15: 1 = Inits/DAs have not been loaded
zero_word	$E1/00BA	2	Two null bytes (guaranteed to be zeros)
OS_Kind	$E1/00BC	1	OS_KIND Indicates currently running operating system, as follows: - $00 = ProDOS 8 - $01 = GS/OS - $FF = none - operating system is being loaded or switched
OS_Boot	$E1/00BD	1	OS_BOOT Indicates the operating system that was initially booted, as follows: - $00 = ProDOS 8 - $01 = GS/OS
OS_Flag	$E1/00BE	2	GSOSBusy Indicates the status of GS/OS, as follows: Bit 15: 0 = GS/OS is not busy 15: 1 = GS/OS is busy Bit 3: PatchedOnceBit if set to 1 Bit 2: PatchLoadedBit if set to 1 Bit 1: BadOSBit if set to 1 Bit 0: SpecialBit if set to 1
	$E1/00FF	2	Deals with INTMGRV - $0000
warm_cold_flag	$E1/01D0	2	Bit 0: 0 = cold startup/shutdown 0: 1 = warm startup/shutdown

GS/OS bank 00 globals

Name	Address	Length	Description
dev_start_flag	$00/B9F0	2	Set to non-zero during device dispatcher startup
fst_tbl_ptr	$00/B9F2	4	Set by SCM to point to FST table
sys_prefs	$00/B9F6	2	Bit 15: 0 = mount procedure will not display mount dialog. It will respond as if user had selected Cancel in mount dialog. 15: 1 = mount procedure will display mount dialog and return user response. Bit 14: 0 = mount dialog will have cancel button. 14: 1 = mount dialog will not have cancel button. Bit 13: 0 = error dialogs (dialogs with only one button) will not be suppressed. 13: 1 = error dialogs will be suppressed. Bits 12 to 0 = reserved for Apple.
char_dev_flag	$00/B9F8	2	Non-zero if get_dnum finds character devices
post_index	$00/B9FA	2	Number of remaining drivers to install
current_speed	$00/B9FC	2	
session_stat	$00/B9FE	2	$0000 : no session in progress $0001 : session in progress

GS/OS event codes

GS/OS can post the following OS events in the IIgs queue:

Name	Value (long)	Description
switch_to_p8	$00000002	Switch from GS/OS to P8
switch_to_gsos	$00000004	Switch from P8 to GS/OS
disk_insert	$00000008	Disk inserted
disk_eject	$00000010	Disk ejected
gsos_shutdown	$00000020	OS shutdown
volume_change	$00000040	Volume changed

GS/OS error codes

Name	Value	Description
no_error	$00	no error has occured
bad_system_call	$01	bad system call number
fst_load_fail	$02	couldn't load FST
invalid_pcount	$04	invalid parameter count
gsos_active	$07	gsos already active
dev_not_found	$10	device not found
invalid_dev_num	$11	invalid device number
drvr_bad_req	$20	bad request or command
drvr_bad_code	$21	bad control or status code
drvr_bad_parm	$22	bad call parameter
drvr_not_open	$23	character device not open
drvr_prior_open	$24	character device already open
irq_table_full	$25	interrupt table full
drvr_no_resrc	$26	resources not available
drvr_io_error	$27	I/O error
drvr_no_dev	$28	device not connected
drvr_busy	$29	driver is busy & not available
drvr_wr_prot	$2B	device is write protected
drvr_bad_count	$2C	invalid byte count
drvr_bad_block	$2D	invalid block number
drvr_disk_sw	$2E	disk has been switched
drvr_off_line	$2F	device off line / no media present
bad_path_syntax	$40	invalid pathname syntax
invalid_ref_num	$43	invalid reference number
path_not_found	$44	subdirectory does not exist
vol_not_found	$45	volume not found
file_not_found	$46	

dup_pathname	$47	create or rename with existing name
volume_full	$48	
vol_dir_full	$49	volume directory full
version_error	$4A	
bad_store_type	$4B	bad storage type
end_of_file	$4C	
out_of_range	$4D	position out of range
invalid_access	$4E	access not allowed
buff_too_small	$4F	buffer too small
softerrorlow	$50	errors from $50 to $6f are soft errors
file_busy	$50	file is already open
dir_error	$51	directory error
unknown_vol	$52	unknown volume type
parm_range_err	$53	parameter out of range
out_of_mem	$54	out of memory
dup_volume	$57	duplicate volume name
not_block_dev	$58	not a block device
invalid_level	$59	specified level outside legal range
damaged_bitmap	$5A	block number too large
bad_path_names	$5B	invalid pathnames for change_path
not_system_file	$5C	not an executable file
os_unsupported	$5D	operating system not supported
stack_overflow	$5F	too many applications on stack
data_unavail	$60	data unavailable
end_of_dir	$61	end of directory has been reached
invalid_class	$62	invalid FST call class
res_not_found	$63	file does not contain req. resource
invalid_fst_id	$64	specified FST is not present in system
invalid_fst_op	$65	FST does not handle this type of call
fst_caution	$66	FST handled call, but result is weird
dup_device	$67	used internally only!!!
dev_list_full	$68	device list is full
sup_list_full	$69	supervisor list is full
fst_error	$6A	generic FST error
softerrorhigh	$6F	maximum soft error number allowed
resource_exist	$70	cannot expand file, resource already exist
res_add_err	$71	cannot add resource fork to this type file
network_error	$88	generic network error

GS/OS Buffers

GS/OS system buffers

FST global buffer	:	`$00/9A00..$00/9DFF`
System buffer	:	`$00/AA00..$00/AC0C`
GQuit stack space	:	`$00/BB00..$00/BCFF`
Direct page	:	`$00/BD00..$00/BDFF`
GS/OS stack space	:	`$00/BE00..$00/BFFF`

GS/OS direct page map

Name	Offset	Length	Comments
Driver equates			
drvr_dev_num	+00	2	deviceNum Device number
drvr_call_num	+02	2	callNum Call number
drvr_buf_ptr	+04	4	bufferPtr Buffer pointer
drvr_slist_ptr	+04	4	bufferPtr Pointer to SIB
drvr_clist_ptr	+04	4	bufferPtr Pointer to control list
boot_slot_equ	+04	2	Boot slot for dispatcher only
dev_id_ref	+04	2	Indirect device ID
dev_char_config	+06	2	BASIC/Pascal 1.1 configuration
dev_pro_config	+08	2	ProDOS configuration
drvr_req_cnt	+08	4	requestCount Request count
drvr_tran_cnt	+0C	4	transferCount Transfer count
drvr_blk_num	+10	4	blockNum Block number
drvr_blk_size	+14	2	blockSize Block size
drvr_fst_num	+16	2	FSTNum Bit 15: 1 = Force device to read block
drvr_stat_code	+16	2	statusCode Status code for status call
drvr_ctrl_code	+16	2	controlCode Control code for control call
drvr_vol_id	+18	2	volumeID Volume ID

drvr_cache	+1A	2	cachePriority Cache priority
drvr_cach_ptr	+1C	4	cachePointer Pointer to cached block
drvr_dib_ptr	+20	4	dibPointer Pointer to active DIB
drvr_dev_ptr	+24	4	Pointer to device list
gen_drvr_ptr	+28	4	Pointer to generated driver list
fw_addr	+28	4	Pointer to current slot

Device dispatcher file list structure

file_list_ptr	+2C	4	Pointer to GS/OS file name list

GS/OS equates

call_number	+30	2	FST call number
param_blk_ptr	+32	4	Pointer to user's parameter block
dev_num	+36	2	Device number from parameter block
dev1_num	+36	2	Alias name for dev_num
dev2_num	+38	2	Second device number
path1_ptr	+3A	4	Pointer to 1st partial/entire pathname
fcr_ptr	+3A	4	Pointer to file control record
path2_ptr	+3E	4	Pointer to 2nd partial/entire pathname
vcr_ptr	+3E	4	Pointer to volume control record

path_flag	+42	2	Flag for path information
			`Bit 14: 0 = pathname1 is null`
			`14: 1 = pathname1 is non-null`
			`Bit 06: 0 = pathname2 is null`
			`06: 1 = pathname2 is non-null`

span1	+44	2	Largest distance between path1 terms
span2	+46	2	Max distance between separators for path2

SCM equates

segment_table	+4C	4	Pointer to segment table
size	+50	2	Size of segment table

Device dispatcher file list structure

file_entry_ptr	+56	4	Pointer to file list entry

Cache Manager direct page

cache_cur_ptr	+5A	4	Pointer to current cache cell
cache_nxt_ptr	+5E	4	Pointer to next cache cell
cache_pre_ptr	+62	4	Pointer to previous cache cell
cache_bkt_ptr	+66	4	Pointer to bucket list

Supervisory driver direct page

post_drvr_tbl	+6C	4	Used for dynamic driver installation
sup_drvr_ptr	+70	4	Pointer to supervisory driver list

sib_ptr	+74	4	Pointer to SIB
sup_parm_ptr	+78	4	Pointer to supervisory parameters

FST direct page

FST space	+$80..$D3	54

SCM temporary direct page usage

Ptr	+E8	4
M_temp	+EC	4
Seg	+F0	4
Vp	+F4	4
Vcr	+F8	4
Hand	+FC	4
Pb_ptr	+FC	4

System service calls

About

Access to several system service routines has been provided for File System Translators and Device Drivers by GS/OS. Access to these routines is through a System Service Dispatch Table located in bank $01 from $FC00 to $FCFF.

System Service Dispatch Table

System Service name	Address	Location
DEV_DISPATCHER	$02/15EF	$01/FC00
CACHE_FIND_BLK	$00/A3B1	$01/FC04
CACHE_ADD_BLK	$00/A32E	$01/FC08
CACHE_INIT	$00/A2D6	$01/FC0C
CACHE_SHUTDN	$00/A652	$01/FC10
CACHE_DEL_BLK	$00/A422	$01/FC14
CACHE_DEL_VOL	$00/A51F	$01/FC18
ALLOC_SEG	$00/FCE8	$01/FC1C
RELEASE_SEG	$00/FD15	$01/FC20
ALLOC_VCR	$00/F59F	$01/FC24
RELEASE_VCR	$00/F6ED	$01/FC28
ALLOC_FCR	$00/F5A2	$01/FC2C
RELEASE_FCR	$00/F72B	$01/FC30
SWAP_OUT	$00/F997	$01/FC34
DEREF	$00/FE09	$01/FC38
GET_SYS_GBUF	$00/E26C	$01/FC3C
SYS_EXIT	$00/B7C7	$01/FC40
SYS_DEATH	$00/EF02	$01/FC44
FIND_VCR	$00/F863	$01/FC48
FIND_FCR	$00/F872	$01/FC4C
SET_SYS_SPEED	$02/1D5E	$01/FC50
CACHE_FLSH_DEF	$00/A5FE	$01/FC54
RENAME_VCR	$00/F763	$01/FC58
RENAME_FCR	$00/F766	$01/FC5C
GET_VCR	$00/F93C	$01/FC60
GET_FCR	$00/F944	$01/FC64
LOCK_MEM	$00/FE35	$01/FC68
UNLOCK_MEM	$00/FE25	$01/FC6C
MOVE_INFO	$00/A05B	$01/FC70
CVT_0TO1	$00/E49A	$01/FC74
CVT_1TO0	$00/E4D0	$01/FC78
REPLACE_80	$00/E50B	$01/FC7C
TO_B0_CORE	$E0/FC5A	$01/FC80
GEN_DISPATCH	$E0/F258	$01/FC84
SIGNAL	$00/ECFE	$01/FC88
GET_B0_BUFF	$E0/FCC8	$01/FC8C
SET_DISKSW	$02/1825	$01/FC90

REPORT_ERROR	$00/EFD8	$01/FC94
MOUNT_MESSAGE	$E1/DF11	$01/FC98
FULL_ERROR	$00/F04C	$01/FC9C
REPORT_FATAL	$00/EF8E	$01/FCA0
SUP_DRVR_DISP	$E0/FCCD	$01/FCA4
INSTALL_DRIVER	$E0/FE5B	$01/FCA8
GET_BOOT_PFX	$00/E466	$01/FCAC
SET_BOOT_PFX	$00/E471	$01/FCB0
ALLOC_CACHE_SEG	$00/FB7C	$01/FCB4
GET_STKED_ID	$E1/D27B	$01/FCB8
DYN_SLOT_ARBITER	$02/1D09	$01/FCBC
PARSE_PATHNAME	$00/B6BE	$01/FCC0
POST_OS_EVENT	$00/B300	$01/FCC4
DYNAMIC_INSTALL	$E0/FD48	$01/FCC8
DEV_MGR_SVC	$02/18DA	$01/FCCC
OLD_DEV_DISP	$E0/E1FD	$01/FCD0
INIT_PARSE_PATH	$00/B6AF	$01/FCD4
UNBIND_INT_VECT	$00/EB37	$01/FCD8
DO_INSERT_SCAN	$E1/DE64	$01/FCDC
TOOLBOX_MSG	$00/F03B	$01/FCE0
not used		$01/FCE4
not used		$01/FCE8
not used		$01/FCEC
not used		$01/FCF0
not used		$01/FCF4
not used		$01/FCF8
RTL		$01/FCFC

Making a System service call from an application

You can call system service calls from an application if the right settings have been applied either to the direct page or to the registers (e.g. AXY):

```
Entry LDAL  GSOSBusy
      BMI   Exit
      ORA   #$8000
      STAL  GSOSBusy

      PHD
      LDA   #$BD00
      TCD
      SEP   #$20
      LDAL  $E0C068
      PHA
      LDAL  $E0C08B
      LDAL  $E0C08B
      REP   #$20

      ...set GS/OS direct page parameters
      ...call your system service call

      SEP   #$20
      PLA
      STAL  $E0C068
      REP   #$20
      PLD

      LDAL  GSOSBusy
      ASL
      LSR
      STAL  GSOSBusy
Exit  RTS
```

System service calls description

ALLOC_CACHE_SEG ($01/FCB4)

Description	This routine allocates a block of memory and returns a virtual pointer to it.
Parameters	Input: A register: requested size Return: A register: error code if carry set X register: low address to allocated virtual pointer Y register: high address to allocated virtual pointer
Notes	The maximum memory block size that can be requested is $1B00-byte long.
Errors	If c=0: successful call If c=1: out of memory or bad parameter (requested size)

ALLOC_FCR ($01/FC2C)

Description	This routine returns a virtual pointer to a File Control Record of the requested size.
Parameters	Input: A register: requested memory block size (number of bytes) X register: pointer (low byte) to class 1 input string of file name Y register: pointer (high byte) to class 1 input string of file name Return: X register: virtual pointer (low byte) to newly allocated block Y register: virtual pointer (high byte) to newly allocated block
Notes	None.
Errors	If c=0: no error; memory was allocated. If c=1: error; memory could not be allocated.

ALLOC_SEG ($01/FC1F)

Description	This routine returns a virtual pointer to a segment of the requested size.
Parameters	Input: A register: requested memory block size (number of bytes) Return: X register: virtual pointer (low byte) to newly allocated block Y register: virtual pointer (high byte) to newly allocated block
Notes	None.
Errors	If c=0: no error; memory was allocated. If c=1: error; memory could not be allocated.

ALLOC_VCR ($01/FC24)

Description	This routine returns a virtual pointer to a Volume Control Record of the requested size.
Parameters	Input: A register: requested memory block size (number of bytes) X register: pointer (low byte) to class 1 input string of volume name Y register: pointer (high byte) to class 1 input string of volume name Return: X register: virtual pointer (low byte) to newly allocated block Y register: virtual pointer (high byte) to newly allocated block
Notes	None.
Errors	If c=0: no error; memory was allocated. If c=1: error; memory could not be allocated.

CACHE_ADD_BLK ($01/FC08)

Description

This routine attemps to add the requested block to the cache. The block is added at the start of the LRU chain (that is, at those most recently used). If there is not enough room in the cache, the block(s) at the end of the chain (that is, at those least recently used) are purged until there is enough room for the requested block.

Parameters

Input:
GS/OS direct page:
- blockSize
- blockNum
- deviceNum
- volumeID
- cachePriority

Return:
GS/OS direct page:
- cachePointer — Pointer to start of block in cache

Notes

Full native mode is always assumed.

When drivers make this call, the block is cached by the device number.

Errors

If c=0: no error; block was added to the cache.
If c=1: error; block was not added to the cache.

CACHE_DEL_BLK ($01/FC14)

Description	This routine attemps to delete the specified block from cache memory.
Parameters	Input: GS/OS direct page: blockSize blockNum deviceNum volumeID cachePriority Return: None
Notes	Input and output are always passed by GS/OS direct-page locations in this routine. Full native mode is used.
Errors	If c=0: no error; block was deleted from cache. If c=1: error; block was not deleted from cache.

CACHE_DEL_VOL ($01/FC18)

Description	This routine will try to delete all blocks belonging to the requested device number from the cache. If the device number = 0 then all blocks of all device numbers of the specified FST will be deleted. If both device number and FST number are zero then all blocks that are not deferred will be deleted.
Parameters	Input: GS/OS direct page: deviceNum FSTNum Return: None
Notes	None.
Errors	If c=0, no error; the volume's block(s) are deleted. If c=1, an error occurred; the blocks may still be cached.

CACHE_FIND_BLK ($01/FC04)

Description

This routine attemps to find the requested block in the cache. If the block is found, it is moved to the start of the LRU chain, and a 4-byte pointer to its start is returned to the caller. One of two possible searches may be specified for this call: a search by device number (used by drivers) or a search by volume ID (used by FSTs when a deferred-write session is in progress). A routine making this system service call must specify the type of search desired by setting the carry flag appropriately.

Parameters

Input:
GS/OS direct page:
- blockNum
- deviceNum
- volumeID

Carry flag : 0 = search by device number
1 = search by volume ID

Return:
GS/OS direct page:
- cachePointer Pointer to start of block in cache

Notes

Full native mode is always assumed.
Drivers making this call should request a search by device number (c=0).

Errors

If c=0: no error; block is in cache.
If c=1: error; block is not in cache.

CACHE_INIT ($01/FC0C)

Description	This routine will try to initialize the cache. Memory as needed by the cache is obtained from the Memory Manager.
Parameters	Input: None Return: None
Notes	The size of the cache is determined by looking at battery ram. Once this is read, changing the value in battery ram will not change the size of the cache. The cache size cannot be changed on the fly.
Errors	If c=0, no error. If c=1, an error occurred.

CACHE_FLSH_DEF ($01/FC54)

Description	This routine will try to write to disk the deferred cache blocks belonging to a specific volume id. After being written to disk, these deferred cache blocks will be downgraded to regular cache blocks.
Parameters	Input: None Return: None
Notes	Input and output is passed to this routine by GS/OS direct page and full native mode is always assumed.
Errors	If c=0, no error; the volume ids deferred blocks have been written to disk. If c=1, an error occurred; couldn't write to disk all deferred blocks of the given volume id.

CACHE_SHUTDN ($01/FC10)

Description	This routine will try to shutdown the cache by deleting each entry one at a time. The LRU list will be used for deletion, the bucket lists will not be used nor updated. The state of the cache is unknown if there's an error.
Parameters	Input: None Return: None
Notes	None.
Errors	If c=0: no error; the cache has been shutdown. If c=1: error; the cache is unreliable now.

CVT_0TO1 ($01/FC74)

Description	This routine converts a class 0 string into a class 1 string.
Parameters	Input: Parameters on stack: Longword pointer — pointer to source string Longword pointer — pointer to destination string Return: Class 0 source string converted to class 1 destination string
Notes	Source and destination pointers may be identical.
Errors	None.

CVT_1TO0 ($01/FC78)

Description	This routine converts a class 1 string into a class 0 string.
Parameters	Input: Parameters on stack: Longword pointer — pointer to source string Longword pointer — pointer to destination string Return: Class 1 source string converted to class 0 destination string
Notes	Source and destination pointers may be identical.
Errors	If c=0: successful call. If c=1: source string length > 255.

DEREF ($01/FC38)

Description	This routine dereferences a virtual pointer and returns a pointer corresponding to the current location of the block referenced by the virtual pointer. This is the only way you should dereference virtual pointers.
Parameters	Input: X register: virtual pointer (low byte) Y register: virtual pointer (high byte) Return: X register: pointer (low byte) to dereferenced block Y register: pointer (high byte) to dereferenced block
Notes	The 32-bit pointer return in the X and Y registers points to the first byte in the block.
Errors	None.

DEV_DISPATCHER ($01/FC00)

Description	This routine passes control to a device to execute a command.
Parameters	Input: GS/OS direct page: deviceNum callNum bufferPtr requestCount transferCount blockNum blockSize fstNum volumeID cachePriority cachePointer dibPointer Return: A register: error code
Notes	Input and output are always passed by GS/OS direct-page locations in this routine. Full native mode is used.
Errors	If c=0: no error; the command has been executed. If c=1: error; the command has not been executed.

DEV_MGR_SVC ($01/FCCC)

Description	This is the main entry point for the device manager. It is responsible for parsing the command and dispatching to the appropriate function.
Parameters	Input: None Return: None
Notes	The following calls are handled: $002C = Class 0 D_INFO $202C = Class 1 D_INFO $202D = Class 1 D_STATUS $202E = Class 1 D_CONTROL $202F = Class 1 D_READ $2030 = Class 1 D_WRITE $2036 = Class 1 D_RENAME
Errors	If c=0, no error. If c=1, an error occurred.

DO_INSERT_SCAN ($01/FCDC)

Description	System Service Vector handler which allows a module external to the OS to perform a device scan looking for a disk insertion. This procedure saves the callers DBR and DP registers, sets these registers to those needed by the OS routines, and then performs the device scan.
Parameters	Input: None Return: A register: device number of the first device that had a disk inserted, if the carry is clear
Notes	On entry, it is assumed that the language card memory is banked in properly (otherwise, it won't ever get here), and that the processor is in full native mode.
Errors	If c=0: disk inserted. If c=1: no insertion was seen.

DYN_SLOT_ARBITER ($01/FCBC)

Description

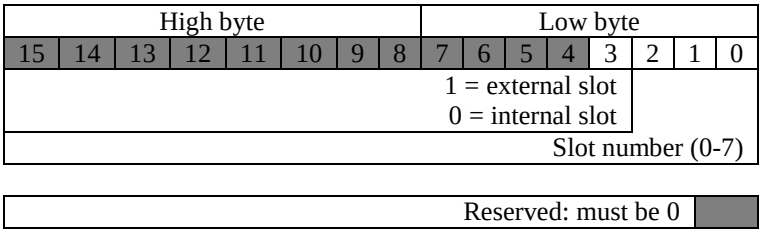
This call might provide support for dynamic switching between devices on internal and external slots in the future. At the time of publication, the call indicates whether the slot is available.

Parameters

Input:
A register: requested slot

Return:
X register: byte-encoded slot configuration
Carry flag: cleared if requested slot was granted
Set if requested slot was denied

Requested slot: word input value: specifies the slot to be requested. The requested-slot parameter has this format:



Notes

None.

Errors

Carry flag set if request denied.

DYNAMIC_INSTALL ($01/FCC8)

Description	This system service routine installs into the device list the driver whose DIB address is passed in the X and Y registers. Then a startup call is issued to the new device.
Parameters	Input: X register: DIB address (low word) Y register: DIB address (high word) Return: None
Notes	If the device returns an error it will be purged from the device list. DYNAMIC_INSTALL uses a pointer to a DIB when INSTALL_DRIVER uses a pointer to a list of DIBs.
Errors	None.

FIND_FCR ($01/FC4C)

Description	This routine attempts to find the requested File Control Record.
Parameters	Input: A register: reference number or $0000 X register: pointer (low byte) to class 1 input string of file name (if Acc = $0000) Y register: pointer (high byte) to class 1 input string of file name (if Acc = $0000) Return: X register: virtual pointer (low byte) to File Control Record Y register: virtual pointer (high byte) to File Control Record
Notes	That call is case insensitive.
Errors	If c=0: no error; the File Control Record has been found. If c=1: error; the File Control Record has not been found.

FIND_VCR ($01/FC48)

Description	This routine attempts to find the requested Volume Control Record.
Parameters	Input: A register: volume ID or $0000 X register: pointer (low byte) to class 1 input string of volume name (if Acc = $0000) Y register: pointer (high byte) to class 1 input string of volume name (if Acc = $0000) Return: X register: virtual pointer (low byte) to Volume Control Record Y register: virtual pointer (high byte) to Volume Control Record
Notes	That call is case insensitive.
Errors	If c=0: no error; the Volume Control Record has been found. If c=1: error; the Volume Control Record has not been found.

FULL_ERROR ($01/FC9C)

Description	This routine displays an internationalized error message and returns the pressed button.
Parameters	Input: Parameters on stack: Word value — error message number Longword pointer — table of pointers to substitution strings Return: A register: result code indicating which button was pressed
Notes	None.
Errors	If c=0; no error. If c=1; an error has occurred.

GEN_DISPATCH ($01/FC84)

Description	This routine is the central dispatcher for generated driver calls. This routine parses the command number and dispatches control to the device driver via the proper generated driver core routine. The driver block number and buffer address are preserved throughout the driver call by the driver dispatcher.
Parameters	Input: A register: call number Return: A register: error code if carry set
Notes	None.
Errors	If c=0: no error. If c=1: an error occurred.

GET_B0_BUFF ($01/FC8C)

Description	This routine returns the address of the bank 0 system buffer.
Parameters	Input: None Return: X register: address of bank 0 system buffer
Notes	None.
Errors	None.

GET_BOOT_PFX ($01/FCAC)

Description	This routine returns the boot prefix, known as prefix #32 or "*".
Parameters	Input: None Return: X register: low pointer to class 1 boot prefix Y register: high pointer to class 1 boot prefix
Notes	None.
Errors	If c=0: successful call. If c=1: the prefix length is null.

GET_FCR ($01/FC64)

Description	This routine returns the requested File Control Record that was allocated with the ALLOC_FCR call. The accumulator value refers to a record's relative position in the FCR list rather than an FCR value.
Parameters	Input: A register: record index (or $0000 for next record) Return: X register: virtual pointer (low byte) to File Control Record Y register: virtual pointer (high byte) to File Control Record
Notes	None.
Errors	If c=0: no error; the File Control Record has been found. If c=1: error; the File Control Record has not been found.

GET_STKED_ID ($01/FCB8)

Description	This routine returns the ID of the next item on the ID stack.
Parameters	Input: None Return: A register: ID of next item on stack, 0 if stack empty
Notes	The 64-byte long stack ID contains the ID of programs to return to.
Errors	None.

GET_SYS_GBUF ($01/FC3C)

Description	This routine returns a pointer to the GS/OS global buffer for FST use.
Parameters	Input: None Return: X register: pointer (low byte) to GS/OS global buffer for FST use Y register: pointer (high byte) to GS/OS global buffer for FST use
Notes	Full native mode is used. The carry is always cleared. At that time, gbuf is located at $00/9A00 and is $400 bytes long.
Errors	None.

GET_VCR ($01/FC60)

Description	This routine returns the requested Volume Control Record that was allocated with the ALLOC_VCR call. The accumulator value refers to a record's relative position in the VCR list rather than an VCR value.
Parameters	Input: A register: record index (or $0000 for next record) Return: X register: virtual pointer (low byte) to Volume Control Record Y register: virtual pointer (high byte) to Volume Control Record
Notes	None.
Errors	If c=0: no error; the Volume Control Record has been found. If c=1: error; the Volume Control Record has not been found.

INIT_PARSE_PATH ($01/FCD4)

Description	This routine clears internal variables for parse_path call.
Parameters	Input: Carry flag: = 0: do initialization = 1: call release_vptrs Return: None
Notes	The release_vptrs routine releases space occupied by strings pointed to by virtual pointers v_ptr1 and v_ptr2.
Errors	None.

INSTALL_DRIVER ($01/FCA8)

Description	Because GS/OS supports removable, partitionable media on block devices, it must be able to install devices dynamically in its device list as new partitions come on line. INSTALL_DRIVER has been provided for that purpose. Important: The existence of this call implies that the GS/OS device list can grow during execution. Drivers and applications cannot count on a fixed device list. See “Scanning the Device list,” in Chapter 11, “System Service Calls”, of GS/OS Driver reference.
Parameters	Input: X register: DIB list address (low word) Y: register: DIB list address (high word) Return: A register: error code DIB list address: longword input pointer: specifies the address of a list of device information blocks to be installed into the device list. The first field in the list is a longword that specifies the number of device information blocks to be installed; it is followed by a series of longword pointers, one to each DIB to be installed.
Notes	This system service routine posts a DIB address for future driver installation. Installation will occur following the current or next return from the device dispatcher. This call informs the device dispatcher that adriver or set of drivers is to be dynamically installed into the device list at the end of the next time the driver returns to the device dispatcher. When installing the driver, the device dispatcher inserts the device into the device list and then issues a startup call to the device. If space cannot be allocated in the device list for the new device, or if the device returns an error as a result of the startup call, the n the device will not be installed into the device list.
Scanning the device list	There is no indication to an application that the device list has changed size as a result of this call. An application (such as the Finder) that scans block devices should always begin by issuing a DInfo call to device $0001 and should continue up the device list until error $11 (invalid device number) occurs. The DInfo call should have a parameter count of $0003 to give the application each device’s device-characteristics word. If the new device is a block device with removable media, the application should make a status call to the device. If applications scan devices in this manner, dynamically installed devices will always be included in the scan operation.
Errors	Error checking is critical when using this call. Two possible errors may be returned. If error $54 (out of memory) occurs, it is not possible to install any drivers; if error $29 (device busy) occurs, it means that an INSTALL_DRIVER is already pending. In case the latter current driver installation cannot be accepted, the device driver must wait until it is accessed once more before it can install additional devices.

LOCK_MEM ($01/FC68)

Description	This routine locks all memory segments that were allocated with the ALLOC_SEG call. Use UNLOCK_MEM when you no longer need these segments; otherwise, the system could run of available memory.
Parameters	Input: None Result: None
Notes	This routine always locks the segments. It counts the number of times that a call is made so that Unlock_mem will only unlock the segments when an equal number of locks and unlocks are done. This allows nested lock/unlocks to work.
Errors	None.

MOUNT_MESSAGE ($01/FC98)

Description	This routine displays an internationalized mount volume message and returns the pressed button.
Parameters	Input: A register: display message type Parameters on stack: Long word pointer Pointer to volume name Return: A register: result code indicating which button was pressed.
Notes	The mount message is displayed only under certain conditions depending on the Acc value: $0000: the appearance of the box is dictated by the App preference word (msb) which is set or reset by the Set_Sys_Prefs call. $0001: the FST can mandate the appearance of the message
Errors	If c=0: no error. If c=1: an error has occurred.

MOVE_INFO ($01/FC70)

Description

This call transfers a block of data from a source buffer to a destination buffer. MOVE_INFO can be used by device drivers to transfer data from a single I/O location to a buffer or from a buffer to a single I/O location.

Parameters

The source buffer pointer, destination buffer pointer, and number of bytes to transfer are passed as input parameters to this routine via the stack. Source and destination buffers may be in the same or different memory banks, and either may straddle a bank boundary.

Input:
This is how the stack looks on entry to the call (before execution of the JSL instruction):

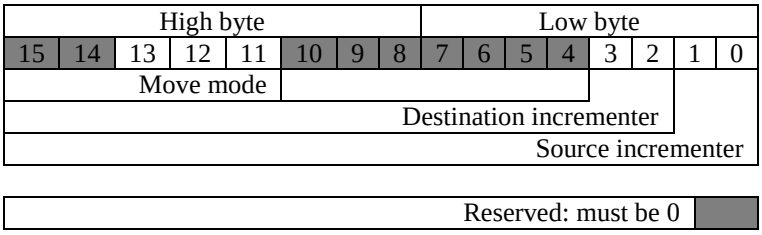
Parameters on stack	**Size and type**	**Description**
	<- stack pointer	
previousContents		
sourcePtr	Longword pointer	Pointer to source buffer
destinationPtr	Longword pointer	Pointer to destination buffer
transferCount	Longword value	Number of bytes to transfer
commandWord	Word value	Flags (see description below)

The high bytes of sourcePtr, destinationPtr, and transferCount must be 0.

Return:
Data Bank register: unchanged
Direct register: unchanged
Accumulator: error code
X register: undefined
Y register: undefined

Command word

The command word tells MOVE_INFO what kind of transfer to make and how to increment the destination and source address (useful, for example, for inverting the order of data as it is copied or for filling memory with a single value). The command format is this:



Where move mode can have these values and meanings:

000	(Reserved)
001	Block move
010-111	(Reserved)

and destination incrementer ca nhave these values and meanings:

00	Constant destination
01	Increment destination by 1
10	Decrement destination by 1
11	(Reserved)

and source incrementer can have these values and meanings :

00	Constant source
01	Increment source by 1
10	Decrement source by 1
11	(Reserved)

Presently, only block moves are defined.

Source incrementer and destination incrementer define in what order successive bytes are transferred from the source buffer and in what order they are placed in the destination buffer. The following recommended predefined constant values for the MOVE_INFO command word cover most typical situations:

```
Move mode:
moveblkcmd              equ     $0800
                        (a block move)
Most common command:
move_sinc_dinc          equ     $05+moveblkcmd
                        (source and destination both increment)
Less common commands:
move_sinc_dinc          equ     $0+moveblkcmd
                        (source increments, destination decrements)
move_sdec_dinc          equ     $0+moveblkcmd
                        (source decrements, destination increments)
move_sdec_ddec          equ     $0+moveblkcmd
                        (source decrements, destination decrements)
move_scon_dcon          equ     $0+moveblkcmd
                        (source constant, destination constant)
move_sinc_dcon          equ     $0+moveblkcmd
                        (source increments, destination constant)
move_sdec_dcon          equ     $0+moveblkcmd
                        (source decrements, destination constant)
move_scon_dinc          equ     $0+moveblkcmd
                        (source constant, destination increments)
move_scon_ddec          equ     $0+moveblkcmd
```

(source constant, destination decrements)

With these various combinations, buffers can be emptied or filled from the bottom up or from the top down, and single values can be placed in a buffer from the bottom up or from the top down. Some of the values are particularly helpful for moving data from one buffer into another buffer that overlaps the first.

Calling sequence

From assembly language, you set up and invoke MOVE_INFO like this:

1. Place machine in full native mode (e=0, m=0, x=0).
2. Push parameters onto stack as shown under “Parameter,” earlier in this section.
3. Execute this instruction:
   jsl Move_Info

Sample code

Here is an assembly-language example of a call to MOVE_INFO:

```
rep     #$30
pea     source_pointer|-16      ; source pointer
pea     source_pointer
pea     dest_pointer|-16        ; destination pointer
pea     dest_pointer
pea     count_length|-16        ; count length
pea     count_length
pea     move_sinc_dinc          ; command word
jsl     move_info
```

Errors

If c=0: no error.
If c=1: error.

OLD_DEV_DISP ($01/FCD0)

Description	This is the main entry point for the device dispatcher.
Parameters	Input: None Return: A register: error code if carry set
Notes	Calls to device zero specify calls to the dispatcher and not to a particular device. Non-zero device numbers will be passed on to that specific device.
Errors	If c=0: no error. If c=1: an error occurred.

PARSE_PATHNAME ($01/FCC0)

Description	This routine translates pathname into canonical format for FST_Specific call.
Parameters	Input: A register: offset to pathname pointer in parameter list X register: FST attribute flags word from FST header Y register: $0000 to select pathname1 $0001 to select pathname2. Return: A register: error code if error GS/OS direct page path_flag: indicates which pathname active. dev1_num: device number if present and path1 active. dev2_num: device number if present and path2 active. span1: maximum name length if path1 active. span2: maximum name length if path2 active. path1_ptr: pointer to pathname if path1 active. path2_ptr: pointer to pathname if path2 active.
Notes	Must call system service call init_parse_path once before calling this routine.
Errors	If c=0: no error. If c=1: an error has occurred.

POST_OS_EVENT ($01/FCC4)

Description	This routine posts an OS event in the event queue.
Parameters	Input: A register: low word of event code X register: high word of event code Parameters on stack: Word values Two words are pushed Return: None
Notes	None.
Errors	If c=0: no error; the event has been posted. If c=1: error; the event has not been posted.

RELEASE_FCR ($01/FC30)

Description	This routine releases a memory File Control Record that was allocated with the ALLOC_FCR call.
Parameters	Input: A register: File Control Record reference number Return: None
Notes	None.
Errors	If c=0: no error; memory was freed. If c=1: error; memory was not freed.

RELEASE_SEG ($01/FC20)

Description	Releases a memory segment that was allocated with the ALLOC_SEG call.
Parameters	Input: X register: virtual pointer (low byte) to target block Y register: virtual pointer (high byte) to target block Return: None
Notes	None.
Errors	If c=0: no error; memory was freed. If c=1: error; memory was not freed.

RELEASE_VCR ($01/FC28)

Description	This routine releases a memory Volume Control Record that was allocated with the ALLOC_VCR call.
Parameters	Input: A register: Volume Control Record reference number Return: None
Notes	None.
Errors	If c=0: no error; memory was freed. If c=1: error; memory was not freed.

RENAME_FCR ($01/FC58)

Description	This routine renames a File Control Record.
Parameters	Input: A register: reference number X register: low pointer to new file name Y register: high pointer to new file name Return: None
Notes	None.
Errors	If c=0: no error. If c=1: error; an out of memory error occurred.

RENAME_VCR ($01/FC5C)

Description	This routine renames a Volume Control Record.
Parameters	Input: A register: volume ID X register: low pointer to new volume name Y register: high pointer to new volume name Return: None
Notes	None.
Errors	If c=0: no error. If c=1: error; an out of memory error occurred.

REPLACE80 ($01/FC7C)

Description

This routine replaces each occurrence of sep (ASCII value $80) in the input string with a character (repl_char) specified by the caller. However, if there are any occurrences of repl_char in the input string, this routine leaves the input string unchanged and returns an error.

Parameters

Input:
Parameters on stack:

Longword pointer	Pointer to a class 1 string
Word value	Word containing replacement character in low byte

Return:
Input string with seps converted to repl_char (or unchanged if input string already contained occurrences of repl_char)

Notes

None.

Errors

If c=0: successful call.
If c=1: error, replacement character already appeared in input string.

REPORT_ERROR ($01/FC94)

Description	This routine displays an internationalized error message with two substitution strings and returns the pressed button.
Parameters	Input: Parameters on stack: Word value — Error number Long word pointer — Pointer to 1st substitution string Longword pointer — Pointer to 2nd substitution string Return: A register: result code indicating which button was pressed.
Notes	None.
Errors	If c=0: no error. If c=1: an error has occurred.

REPORT_FATAL ($01/FCA0)

Description

This routine writes out the internationalized fatal error message and returns the pressed button.

Parameters

Input:
Parameters on stack:

Word value	Fatal error number to be printed in message as substitution string 2
Word value	Message number in error message file
Longword pointer	Pointer to substitution string 1 (may be null)

Return:
A register: result code indicating which button was pressed.

Notes

None.

Errors

If c=0: no error.
If c=1: an error has occurred.

SET_BOOT_PFX ($01/FCB0)

Description	This routine sets the boot prefix, known as prefix #32 or “*”.
Parameters	Input: X register: low pointer to class 1 boot prefix Y register: high pointer to class 1 boot prefix Return: None
Notes	None.
Errors	None.

SET_DISKSW ($01/FC90)

Description	Some device drivers detect volume-off-line or disk-switched conditions through device-specific status calls rather than through returned errors. Such a condition would then not be detected by the device dispatcher on exit from the driver call. In fact, by GS/OS convention, off-line and disk-switched conditions should never be returned as errors from a status call; errors are reserved for conditions in which a call fails, not for passing status information. With the call SET_DISKSW, drivers can specifically request that the disk-switched status (maintained internally by the device dispatcher) be set in this situation. SET_DISKSW, if necessary, removes the device's blocks from the cache and places its volumes off line (if the device dispatcher-maintained disk-switched flag has not already been set). All GS/OS drivers are expected to call SET_DISKSW if they detect a disk-switched or off-line condition as a result of a status call.
Parameters	Input: GS/OS direct page: deviceNum: device number of disk-switched device Return: None
Notes	Full native mode is assumed. Register contents are unspecified on entry and return, except that the Data Bank register and Direct register are unchanged by the call.
Errors	None.

SET_SYS_SPEED ($01/FC50)

Description

This call allows hardware accelerators to stay compatible with device drivers that may have speed-dependent software implementations.

Whenever it dispatches to a driver, the device dispatcher obtains the device driver's speed class from the DIB and issues this system service call to set the system speed. When the driver completes the call, the device dispatcher restores the system speed to what it was before the call.

An accelerator card may intercept this vector and replace the system service call with its own routine, thus maintaining compatibility with GS/OS device drivers.

Parameters

Input:
The A register contains one of these speed settings:

$0000	Apple IIGS normal speed
$0001	Apple IIGS fast speed
$0002	Accelerated speed
$0003	Not speed dependent

Settings from $0004 to $FFFF are not valid.

Return:
The accumulator contains the speed setting that was in effect prior to issuing this system service call.

Notes

None.

Errors

None.

SIGNAL ($01/FC88)

Description	This call announces the occurrence of a specific signal to GS/OS and provides GS/OS with the information needed to execute the proper signal handler (previously installed with the ArmSignal subcall of the Driver_Control call). GS/OS queues this information and uses it when it dispatches to the signal handler. For more information on GS/OS signals and signal handlers, see Chapter 9, “Handling Interrupts and Signals,” of GS/OS reference.
Parameters	Input: A register: signal priority X register: low word of signal-handler address Y register: high word of signal-handler address Return: A register: undefined X register: undefined Y register: undefined Signal priority: priority ranking of the signal, with $0000 being the lowest priority and $FFFF being the highest. Signal-handler address: address of the signal-handler entry point.
Notes	A signal source that makes this call as the result of an interrupt should announce no more than one signal per interrupt to avoid the possibility of overflowing the signal queue.
Errors	None.

SUP_DRVR_DISP ($01/FCA4)

Description

This call is the main entry point to the supervisor dispatcher. It dispatches calls among supervisory drivers. Supervisory drivers provide an interface that gives higher-level device drivers access to hardware.

Supervisory-driver calls can be classified into two groups. Calls with a supervisor number of zero are handled by the supervisor dispatcher; calls with a nonzero supervisor number are passed on to a supervisory driver.

The following calls are handled by the supervisor dispatcher and are not passed on to a supervisory driver:

Call no.	Sup. No.	Function
$0000	$0000	GetSupervisorNumber
$0001	$0000	Set_SIB_Pointer
$0002-$FFFF	$0000	(Reserved)

The following calls are dispatched by the supervisor dispatcher to a supervisory driver:

Call no.	Sup. No.	Function
$0000	(Nonzero)	Supervisor_Startup
$0000	(Nonzero)	Supervisor_Startup
$0000	(Nonzero)	Supervisor_Startup

These subcalls and other supervisory-driver calls are described in detail in Chapter 10, “GS/OS Driver Call Reference.”

Notes

None.

Errors

$28 Device not connected.

SWAP_OUT ($01/FC34)

Description	This routine moves offline any volume in the device specified (a volume is offline if its media is not currently in a device). (Actually, all volumes with the passed device number are marked offline; there should never be more than one volume corresponding to a device number.) A volume associated with the specific device that has no open files is deleted from the system.
Parameters	Input: A register: device number Return: None
Notes	This routine walks the list of active VCRs and marks as swapped out any and all VCRs with open files having the correct dev number. Note that in every legitimate case there will be only one swapped in VCR with the device number passed this routine swaps out any and all volumes corresponding to the passed device number. If a VCR with the correct device number is found with no open files, a release_vcr call is done on that VCR.
Errors	None.

SYS_DEATH ($01/FC44)

Description	This routine writes out a system death message.
Parameters	Input: A register: error code in low byte Return: None
Notes	Does not return to the caller.
Errors	None.

SYS_EXIT ($01/FC40)

Description	This routine returns from the GS/OS environment. It returns to the calling program.
Parameters	Input: A register: error code Return: None
Notes	The previous direct-page address and the stack pointer are restored.
Errors	None.

TO_B0_CORE ($01/FC80)

Description	This routine is used to set up the bank 0 core routine for a dispatch. This routine assumes a GDIB is located contiguous to the DIB for the device being accessed.
Parameters	Input: X register: pointer to dispatcher list B register: dispatcher list bank GS/OS direct page: dibPointer pointer to DIB containing slot & class Return: A register: error code if carry set
Notes	This call can only be called from within a device driver. Note that bank 0 of the language card must be banked in.
Errors	If c=0: no error. If c=1: error.

TOOLBOX_MSG ($01/FCE0)

Description	This routine displays an alert message and returns the pressed button.
Parameters	Input: A register: error message number X register: low pointer to table of error messages (like Error.msg) Y register: high pointer to table of error messages (like Error.msg) Return: A register: result code indicating which button was pressed
Notes	None.
Errors	If c=0: no error. If c=1: an error has occurred.

UNBIND_INT_VECT ($01/FCD8)

Description	This call allows the caller to perform an UnbindInt call when GS/OS is busy (typically during shutdown). There is no system service call to bind an interrupt source. To bind an interrupt, use the BindInt GS/OS call.
Parameters	Input: A register: intNum (from the BindInt GS/OS call) Return: None
Notes	None.
Errors	None.

UNLOCK_MEM ($01/FC6C)

Description	This routine releases all locked segments that were created with the ALLOC_SEG call.
Parameters	Input: None Return: None
Notes	This routine unlocks memory segments when an equal number of locks and unlocks are done. This allows nested lock/unlocks to work.
Errors	None.

GS/OS Records

GS/OS communicates with FSTs through the use of File Control Records and Volume Control Records which keep track of all open files and mounted volumes. FCRs and VCRs are both GS/OS virtual pointers.

Virtual Pointers

Virtual pointer is a 32-bit address where X and Y have a special meaning in the Global Info Manager (GIM) system.

Y is an index in the segment table address
X is an offset within the selected segment table

There are three segment tables.

Y=0, points to the VCR table
Y=1, points to the FCR table
Y=2, points to the ICR table

DEREF is an interesting call to understand the behavior of virtual pointers.

Volume Control Record

A Volume Control Record is a virtual pointer to the GS memory which is allocated through the ALLOC_VCR call. It is an extension of a Memory Manager handle. At that time, a VCR is currently $12 bytes in length.

Name	Offset	Length	Description
vcr_id	+00	2	ID of the VCR set by the ALLOC_VCR call
vcr_name	+02	4	Virtual pointer to the volume name
vcr_status	+06	2	Volume status
vcr_open_cnt	+08	2	Number of open files on volume
vcr_fst_id	+0A	2	Owner ID number
vcr_dev	+0C	2	Last device ID to where volume was last seen
vcr_fst_ptr	+0E	4	Pointer to parent FST

The current status equates are:

Name	Value	Description
vcr_swapped	$4000	Volume is swapped out (1 = true)
vcr_swapped_in	$BFFF	Volume is swapped in
vcr_wr_enable	$2000	Volume has been seen write enabled
vcr_wr_unknown	$DFFF	Volume has not been seen write enabled

File Control Record

A File Control Record is a virtual pointer to the GS memory which is allocated through the ALLOC_FCR call. It is an extension of a Memory Manager handle.

The following table describes the minimum parameters that must be defined in a FCR. It is currently $16 bytes in length.

Name	Offset	Length	Description
fcr_ref_num	+00	2	File reference number, set by the ALLOC_FCR call
fcr_path_name	+02	4	Virtual pointer to file's pathname
fcr_fst_id	+06	2	FST ID of owning FST
fcr_vol_id	+08	2	Volume ID of owning VCR
fcr_level	+0A	2	Level that file was opened with
fcr_newline	+0C	4	Virtual pointer to list of newline characters
fcr_newline_len	+10	2	Length of newline list
fcr_mask	+12	2	New line mask
fcr_access	+14	2	Access used to open file

Interrupt Control Record

An interrupt control record contains all the information needed to execute an interrupt handler.

Interrupt Identification Table

This data structure is a dynamic list in which each entry contains a virtual pointer to an interrupt control record. The entry's index is the interrupt id number corresponding to a specific binding between an interrupt source and its interrupt handler.

There are two ways to get to interrupt control records:

1) each interrupt identification table entry points directly to a single ICR and
2) each ICR is on a linked list of ICRs corresponding to the same firmware interrupt vector.

The format of an ICR is as follows:

Name	Offset	Length	Description
next_avail_vrn	+00	4	VP to next ICR corresponding to specific VRN
past_avail_vrn	+04	4	VP to previous ICR corresponding to specific VRN
vect_disp_base	+08	4	Interrupt Handler Memory Address
vect_ref_num	+0A	2	VRN

Normally, only offsets to handler and vrn fields are needed.

VRN to VI Translation Table

This table holds the information needed to translate a vector reference number (VRN) into an index (not byte offset) to the Vector Table entry corresponding to the VRN.
The table consists of 3-word entries. The first two words of each entry represent a range of legal VRN values. If the input VRN falls between the two values, the output index is the input VRN value minus the value of the third word. Value of $00 in the first word of an entry indicates the current end of table.

```
vrn_to_index
            DC.W   $08,$17,$07          ;map VRNs $8-$17 into indexes $01-$10
loc_1       DC.W   $00,$00,$00          ;reserved for future expansion
            DC.W   $00,$00,$00          ;reserved for future expansion
            DC.W   $00,$00,$00          ;reserved for future expansion
loc_2       DC.W   $00
```

Vector Table

This is a table of headers to linked lists of ICRs. The table is indexed by Vector Index (VI), which is an arbitrary index assigned to each firmware interrupt vector. The Vector to Index Translation Table provides the translation from VRNs to VIs. Each linked list consists of the ICRs for interrupt handlers bound to the particular firmware interrupt vector. This table is actually allocated in the memory controlled by the GIM.

VI	Offset in GIM tables	Corresponding vector P16 interrupt handlers
$00	+0	icr_list
$01	+4	irq.aptalk
$02	+8	irq.serial
$03	+12	irq.scan
$04	+16	irq.sound
$05	+20	irq.vbl
$06	+24	irq.mouse
$07	+28	irq.qtr
$08	+32	irq.kbd
$09	+36	irq.response
$0A	+40	irq.srq
$0B	+44	irq.dskacc
$0C	+48	irq.flush
$0D	+52	irq.micro
$0E	+56	irq.1sec
$0F	+60	irq.ext
$10	+64	irq.other
$11	+68	reserved for expansion
$12	+72	reserved for expansion
$13	+76	reserved for expansion
$14	+80	reserved for expansion
$15	+84	reserved for expansion
$16	+88	reserved for expansion
$17	+92	reserved for expansion
$18	+96	reserved for expansion

Vector Dispatch Table

This is a table that must be located in memory below the language card. GS/OS patches the firmware interrupt vectors to point to entries in this table. Each entry is an instruction sequence of the form:

```
ldx #vi
bra common
```

vi is the vector index corresponding to the interrupt vector and common is the label on common processing code that appears after the other table entries.

The interrupt system initialize constructs the table in managed memory. It puts the address of the table into vect_disp_base and the handle to the table in vect_disp_hand.

```
vect_disp_base      DC.L   00000000     ; base address of Vector Dispatch Table
vect_disp_hand      DC.L   00000000     ; handle of the Vector Dispatch Table

;
; Data used by alloc_interrupt and bind_int
;

vect_ref_num        DC.W   0000         ; vector reference number from bind_int
                                        ; ...call ($0000 for alloc_interrupt)
int_hand_addr       DC.L   00000000     ; handler address from alloc_interrupt
                                        ; ...or bind_int
handled             DC.W   0000         ; when polling interrupt handlers tied
                                        ; ...to a specific VRN, tells whether or
                                        ; ...not the interrupt has been handled
                                        ;      $0000 = not yet handled
                                        ;      $8000 = already handled
```

File System Translators

Header

Name	Offset	Length	Description
Signature	+00	4	'FST '
app_entry	+04	4	Pointer to the application entry routine
sys_entry	+08	4	Pointer to the system entry routine
id	+0C	2	File system ID, refer to next page
s_flags	+0E	2	Attributes (see GS/OS reference)
version	+10	2	Version of the FST
BlockSize	+12	2	Size of a block in bytes
MaxVolSize	+14	4	Maximum volume sizes in blocks
reserved	+18	4	
MaxFileSize	+1C	4	Maximum file size in bytes
reserved	+20	4	
FSTName	+24	var	FST name displayed in the Finder
StartupName	+24+var	18	FST name that appears during a text boot

Present and future FSTs

Name	File system ID	Description
	$0000	Reserved
proDOSFSID	$0001	ProDOS/SOS
dos33FSID	$0002	DOS 3.3
dos32FSID	$0003	DOS 3.2
dos31FSID	$0003	DOS 3.1
appleIIpascalFSID	$0004	Apple II Pascal
mfsFSID	$0005	Macintosh (MFS)
hfsFSID	$0006	Macintosh (HFS)
lisaFSID	$0007	Lisa
appleCPMFSID	$0008	Apple CP/M
charFSID	$0009	Character
msDOSFSID	$000A	MS/DOS
highSierraFSID	$000B	High Sierra
iso9660FSID	$000C	ISO 9660
appleShareFSID	$000D	AppleShare
rdos33FSID	$000E	rDOS 3.3
rdos32FSID	$000F	rDOS 3.2
	$0010-$FFFF	Reserved

Calls handled by FSTs

GS/OS calls can be classified by the art of the operating system that handles them. File calls are handled by FSTs, device calls are handled by the Device Manager, and other calls are handled by the GS/OS Call Manager itself.

Call no.	Call name
$2001	Create
$2002	Destroy
$2004	ChangePath
$2005	SetFileInfo
$2006	GetFileInfo
$2007	JudgeName
$2008	Volume
$200B	ClearBackupBit
$2010	Open
$2012	Read
$2013	Write
$2014	Close
$2015	Flush
$2016	SetMark
$2017	GetMark
$2018	SetEOF
$2019	GetEOF
$201C	GetDirEntry
$2020	GetDevNumber
$2024	Format
$2025	EraseDisk
$2033	FSTSpecific

Note that the ProDOS FST implements the two following class 0 calls:
- READ_BLOCK
- WRITE_BLOCK

FST system entry routine

GS/OS and FSTs communicate in native mode through an internal entry point in the FST header. GS/OS calls the FST system routines with the following values in the registers:

A register : undefined
X register : call number * 2
Y register : class number * 2

Name	Call number	FST call description
startup	$0001 (X = $0002)	GS/OS calls each FST upon warm or cold start. The FST can request for memory or get GS/OS direct page address.
shutdown	$0002 (X = $0004)	When the system is shutdown, GS/OS calls each FST's shutdown routine. The FST can deallocate memory.
sys_remove_vol	$0003 (X = $0006)	That call allows a FST to remove a volume control record.
deferred_flush	$0004 (X = $0008)	Allows the FST to write the cached blocks to disk. That call can be called by the SWAP_OUT system service call or when the stop session call is issued.

Appendix – GS/OS technotes

This and all of the other Apple II Technical Notes have been converted to HTML by Aaron Heiss as a public service to the Apple II community, with permission by Apple Computer, Inc. Any and all trademarks, registered and otherwise, are properties of their owners.

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	cript 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(R) 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(R) 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

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 requires 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, suchas volume names, be sure to use the colon as the separator, because if you donot, 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 filesystem translators, reading from a directory is not always as fast as it isfor 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 unableto 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 todo 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 included with the Resource Manager documentation in the System Software 5.0 documentation.

Further Reference

- GS/OS Reference, Volume 1
- System Software 5.0 documentation (APDA)

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

Revised by Matt Deatherage (January 1991)
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 September 1990: 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

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

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

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 Interrypt 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

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

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 thefile_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 0. (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 thesupplied 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 andEraseDisk 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

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

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
