GS/OS Information Manager Interface

Version 0.03

Overview

The GS/OS Information Manager (GIM) is a set of routines which manages memory and commonly used data objects in the GS/OS environment. Most of the memory required by GS/OS is managed by these routines. All calls are made through the System Services Vector Table, and all should be called in full native (16 bit) mode. Each function ends with an 'RTL' so a corresponding 'JSL' should be done in the call.

Memory Management and Virtual Pointers

Memory for GS/OS is, of course, obtained from the GS Memory Manager. Each piece of GS/OS memory does not have its own handle, however. In order to minimize the number of memory manager handles in the system (too many handles can degrade system performance) Memory is obtained from the GS memory manager in discrete chunks on an 'as needed' basis and these chunks are further subdivided. A subsegment is referred to with a 32 bit 'Virtual' Pointer which is functionally equivalent to a GS Memory Manager 'handle', though the implementation is different.

A Virtual Pointer (VP) is dereferenced by a routine called 'Deref'. All routines which refer to chunks of GS/OS memory pass VPs (just as routines using memory from the GS Memory Manager pass handles) to refer to the chunks. If a routine needs to access the contents of a chunk (sometimes called a memory 'block'), it calls the Deref routine with the VP as input and Deref returns a 4 byte pointer to the contents of the block.

It is important to remember that a VP for a given block is constant up to the point that that block is deallocated, but the physical address associated with it can change any time the GS Memory Manager decides to compact memory. An FST can use the Lock_Mem call to lock down GS/OS managed memory if it wishes to ensure that an address associated with a VP will not change. Of course, such an FST should do an Unlock_Mem call before returning to the System Call Manager so that the GS Memory Manager is free to move GS/OS segments as appropriate.

Allocation/Deallocation Routines

Access to the low level memory management routines are provided to the FST or device driver (or for that matter, to the shell) by the calls alloc_seg, release_seg and deref, reachable through the System Services Vector Table. These routines, along with ones that lock/unlock GS/OS memory are described below.
alloc_seg(request_size) : vp

Function: Returns a Virtual Pointer to a memory block of the requested size
Inputs: Acc -> Requested Memory Block Size (bytes) ($7FF4 max)
Outputs: X <- VP to newly allocated block (low)
Y <- VP to newly allocated block (high)
Errors: Carry <- 1 Exception: 'Could Not Allocate Memory'
Notes: This call allocates a block of memory and returns a virtual pointer to it. If memory could not be obtained the carry is returned set.

release_seg(vp)

Function: Frees a block obtained with the allocate call
Inputs: X -> VP to block to free (low)
Y -> VP to block to free (high)
Outputs: None
Errors: Carry <- 1 Exception: 'Could Not Deallocate Memory'
Notes: This call frees a block of memory pointed to by the virtual pointer passed. If for any reason the memory could not be returned the carry is set.

deref(vp) : pointer

Function: Returns a pointer corresponding to the current location of the block referenced by the virtual pointer.

Inputs: X -> VP of block to dereference (low)
Y -> VP to block to dereference (high)

Outputs: X <- Pointer to dereferenced block (low)
Y <- Pointer to dereferenced block (high)

Errors: None

Notes: This call dereferences the virtual pointer passed. The 32 bit value returned points to the first useable byte in the block.. Don't confuse GIM dereferencing with that associated with the System Memory Manager; they are two different things. A deref gives you a pointer that is good until any system call is made that forces memory to move around (any VCR/FCR/alloc_seg call). If you wish to ensure that a pointer will not change you need to use the lock_mem and unlock_mem calls.

lock_mem()

Function: Locks down GS/OS managed memory

Inputs: None

Outputs: None

Errors: None

Notes: All segments created with alloc_seg, alloc_vcr, and alloc_fcr are locked down. It is expected that a call to unlock_mem will be done before returning to the shell, so that unnecessary memory fragmentation can be avoided.

unlock_mem()

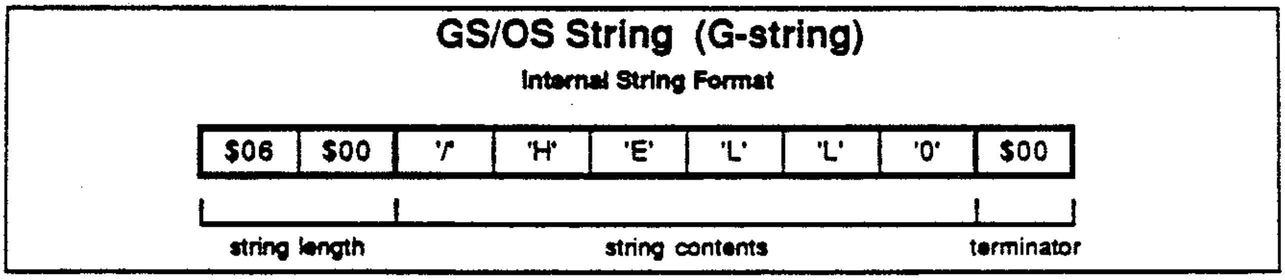
Function: Unlocks GS/OS managed memory

Inputs: None

Outputs: None

Errors: None

Notes: All segments created with alloc_seg, alloc_vcr, and alloc_fcr are unlocked.



GS/OS String (G-string)

Internal String Format

$06	$00	'/'	'H'	'E'	'L'	'L'	'0'	$00

GS/OS Cache Manager ERS

version 0.03

Revision History

0.01	07/21/87	Initial release
0.02	08/19/87	a little updating and removed volume_id
0.03	08/31/87	published as internal and external copy, also added a few more tidbits under implementation notes

Disclaimer

Information contained henceforth is preliminary. Specification and implementation details of GS/OS caching is subject to change without notice.

Introduction

The GS/OS Cache Manager is a set of system level routines used to implement general disk block caching under GS/OS. Briefly, caching is defined here to be the process in which frequently accessed disk blocks are kept in memory in an orderly fashion to facilitate speedy future access to those disk blocks.

As with most caching implementations, the least recently used (LRU) mechanism will be in effect. Caching under GS/OS will be a write through cache. That is, when an FST issues a write call to the device driver, both the block in the cache and the same block on the disk will have the same contents. (Never will the block in the cache contain information more recent than the same block on the disk.) Also, the cache size is user selectable through the control panel.

The cache block size is for all intents and purposes, ***unrestricted*** in size. The GS/OS Cache Manager makes no assumptions about the size of the block to be cached. Cache memory is obtained and released on a as needed basis. If for example 32K is selected as the cache size, then this amount is not directly allocated for specific use by the Cache Manager. This differs from the Mac Cache Manager where it deals only in 512 byte blocks and the cache memory is exclusive to the Cache Manager. (It's true, I asked Bruff about this.)

Description of Cache calls

For each of the following calls, input and output is passed by GS/OS direct page and full native mode is always assumed. Further, on input the B and K registers are don't cares and on output the contents of these registers are preserved. The D register must be set to GS/OS direct page.

The state of the language card is a don't care, the caching routines does not mess with the language card. Carry clear indicates no error, operation complete, otherwise something went wrong.

INIT_CACHE

Before the GS/OS cache can be used, the Cache Manager must be initialized. Initialization will reset the Cache Manager's own internal variables, allocate some memory for it's own housekeeping and determine the size of the user selectable cache. This call is only made by the System Call Manager when GS/OS is booted or reloaded.

ADD_BLOCK

This call will try to add the specified block to the cache. If there is not enough room left in the cache for the specified block, space will be made available by deleting enough cached blocks for this. Blocks will be deleted via LRU. c = 0 means the block was cached. c = 1 means something went wrong and the block wasn't cached.

FIND_BLOCK

This call will traverse the bucket links and will try to find the specified cached block. c = 0 means the block was found. c = 1 means the block wasn't found.

DELETE_BLOCK

This call will traverse the bucket links and will try to release the specified cached block. c = 0 means the block was found and deleted. c = 1 means something went wrong and the block may still be cached.

DELETE_VOLUME

This call will traverse the LRU links and release those cached blocks belonging to the specified device. If the device number is 0 then this call will release those cached blocks belonging to the specified FST. c = 0 means the block(s) or device(s) was deleted. c = 1 means something went wrong and the block(s) or device(s) may still be cached.

SHUTDOWN_CACHE

When the cache is no longer needed, GQUIT will issue this call to shutdown the cache. Shutdown involves releasing back to the system all memory that was allocated to it. c = 0 means cache has been shutdown. c = 1 means something went wrong, the cache may not be shutdown completely. Also, the cache contents are undefined. This call is only called by the System Call Manager.

CACHE_LOCK

This call although specified in the system service table is currently not used at all. Ray Montagne and myself will define this soon. Or we may chuck this call if we can't justify it's existence.

Implementation Notes

At the time of this writing, the caching routines are residing in non bank switched memory. If in the furture the caching routines are moved to bank switched memory, then some GLU must be provided to switch in and out the language cards.

Each cached block will have a header appended to it. The cache header looks like this:

```
c_lru_fwd       gequ   $0000           ;vp to forward LRU vp link
c_lru_bwd       gequ   c_lru_fwd+4     ;vp to backward LRU vp link
c_bkt_fwd       gequ   c_lru_bwd+4     ;vp to forward bucket vp link
c_bkt_bwd       gequ   c_bkt_fwd+4     ;vp to backward bucket vp link
c_blknum        gequ   c_bkt_bwd+2     ;block number of cached block
c_blksize       gequ   c_blknum+4      ;block size of cached block
c_fstnum        gequ   c_blksize+2     ;FST number of cached block
c_devnum        gequ   c_fstnum+2      ;device number of cached block
c_priority      gequ   c_devnum+2      ;cache priority
c_cellsize      gequ   c_priority+2    ;size of this cache cell
c_reserved      gequ   c_cellsize+2    ;reserved by Apple
c_headerlen     gequ   c_reserved+4    ;length of cache header
c_cached_data   gequ   c_reserved+4    ;offset to cached data block
```

The cache list is managed by two sets of doubly linked virtual pointers. One set maintains the least recently used (LRU) order of cached data blocks. The other set maintains these blocks after the block number has been hashed. Currently, the hash function will hash into 128 different buckets. Each bucket contains those blocks that have the same hash value. The hash value for a particular block will always be the same bucket.

For a description of what a virtual pointer is and what it looks like, see the System Call Manager (or Global Information Manger) ERS for details.

The GS/OS cache is limited to the size determined by the user through a battery ram location. There are cases however, where memory cannot be obtained to add a block even though the Cache Manager's own internal variables say there is room. In those cases, the cache will behave as though the cache is full and LRU will be invoked to make room for the add request.

GS/OS System Service Calls

External ERS Ver. 0.11a01

 PRELIMINARY

Revision History

Date	Version	Who	Description of Revision
07-09-1987	0.01	RBM	Initial release with device dispatcher & clr_dev_err.
07-13-1987	0.02	MSA	Added descriptions of alloc_seg, release_seg, deref, alloc_vcr, alloc_fcr, release_vcr, release_fcr, find_vcr, find_fcr, rename_vcr, rename_fcr
07-13-1987	0.03	MSA	Updated to reflect new VCR/FCR structure. Added lock_mem, unlock_mem.
07-14-1987 cache_shutdn,	0:04	RJC	Added descriptions of cache_find_blk, cache_add_blk, cache_init, cache_del_blk, and cache_del_vol.
07-15-1987	0.05	RBM	Added inputs, outputs, errors and notes to device dispatcher & clr_dev_err. Added cache_lock to system service dispatch table. Definition to be provided by RJC ASAP.
07-17-1987	0.06	RJC	Added description of cache_lock.
07-21-1987 get,	0.07	MSA	Added calls get_vcr, get_fcr, and vector locations for rename, lock/unlock, interchange the names get_fcr, find_fcr.
07-23-1987	0.08	MSA	Changed call name clr_dev_err to clr_dev_error.
08-18-1987	0.09	RBM	Changed address' to match current GS/OS memory map. Added call structure for SYS_DEATH.
09-01-1987	0.10	MSA	Changed/shortened and fixed memory management call descriptions
09-02-1987	0.11	MSA	Added description of get_sys_gbuf.
09-02-1987	0.12	JJ	Added description of sys_exit entry point.
09-08-1987	0.13	RBM	Added MOVE_BLOCK routine.
10-08-1987	0.14	MSA	Added CVT_0TO1 and CVT1TO0.
12-22-1987	0.15	RBM	Added: RESERVED_01, RESERVED_02 and SIGNAL
01-06-1988	0.16	RBM	Added: RESERVED_02 and SET_DISKSW
01-08-1988	0.17	RBM	DELETED: CLR_DEV_ERROR
01-08-1988	0.18	FAB	Changed MOVE_BLOCK to MOVE_INFO routine
01-21-1988	0.19	RBM	Added: Set_sys_speed
02-04-1988	0.20	MSA	Change to SWAP_OUT
2-24-1988	0.03a03	RBM	Added: Supervisory driver dispatcher Added: Post driver installation
2-29-1988	0.04a01	JJ	Changed: Description of SYS_DEATH, REPLACE_80 Removed: FULL_ERROR and REPORT_FATAL, which are an internal calls.

6-2-1988	0.08a01	RBM	Added support for linked devices when maintaining device disk switched occurrence. See SET_DISK_SW ($01FC90).
6-21-1988	0.10a01	RBM	Added system service call $01FCBC for dynamic slot arbitration.
7-28-1988	0.11a01	MSA	Added description of GET_BOOT_PFX and SET_BOOT_PFX

About the System Service Calls

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 through $FCFF. A list of these system service routines and their location within the System Service Dispatch Table is shown below:

DEV_DISPATCHER	$01FC00
CACHE_FIND_BLK	$01FC04
CACHE_ADD_BLK	$01FC08
CACHE_INIT	$01FC0C
CACHE_SHUTDN	$01FC10
CACHE_DEL_BLK	$01FC14
CACHE_DEL_VOL	$01FC18
ALLOC_SEG	$01FC1C
RELEASE_SEG	$01FC20
ALLOC_VCR	$01FC24
RELEASE_VCR	$01FC28
ALLOC_FCR	$01FC2C
RELEASE_FCR	$01FC30
SWAP_OUT	$01FC34
DEREF	$01FC38
GET_SYS_GBUF	$01FC3C
SYS_EXIT	$01FC40
SYS_DEATH	$01FC44
FIND_VCR	$01FC48
FIND_FCR	$01FC4C
SET_SYS_SPEED	$01FC50
CACHE_LOCK	$01FC54
RENAME_VCR	$01FC58
RENAME_FCR	$01FC5C
GET_VCR	$01FC60
GET_FCR	$01FC64
LOCK_MEM	$01FC68
UNLOCK_MEM	$01FC6C
MOVE_INFO	$01FC70
CVT_0TO1	$01FC74
CVT_1TO0	$01FC78
REPLACE80	$01FC7C
RESERVED_01	$01FC80
RESERVED_02	$01FC84
SIGNAL	$01FC88
RESERVED_03	$01FC8C
SET_DISKSW	$01FC90
RESERVED_04	$01FC94
RESERVED_05	$01FC98
RESERVED_06	$01FC9C
RESERVED_07	$01FCA0
SUP_DRVR_DISP	$01FCA4
INSTALL_DRIVER	$01FCA8
GET_BOOT_PFX	$01FCAC
SET_BOOT_PFX	$01FCB0
[illegible]	$01FCB8

DYN_SLOT_ARBITER	$01FCBC
RESERVED	$01FCC0 - $01FCFF

Descriptions of each of the system service routines follow:

DEV_DISPATCHER $01FC00

Function: This system service entry point provides access to the device dispatcher. The device dispatcher is responsible for maintenance of the device drivers and also provides the mechanism for dispatching to the device drivers. This system service routine is unique in that one of several system services are provided through this entry point. The actual service provided is specified by the input parameters passed on GS/OS direct page. Services provided through this entry point fall into two classes. Service may pertain to the device dispatcher itself or a specific device driver. A list of the services provided by this entry point is show below:

<u>DEVICE DRIVER</u>

DRVR_OPEN
DRVR_READ
DRVR_WRITE
DRVR_CLOSE
DRVR_STATUS
DRVR_CONTROL
DRVR_FLUSH

Brief descriptions of each of these services are provided in this document. For more detailed information on these services see the Device Dispatcher ERS or the Device Driver ERS.

Drvr_Open

Function: This call is used to prepare a character device for conducting I/O transactions. This may include allocation of resources such as memory for buffers. Block devices will take no action on this call and should return a 'BADCMD' error. Prior to dispatching to the device, the device dispatcher will check that the DIB for the device specified by the device number indicates that the device is a character device.

Inputs: GS/OS Direct Page

Outputs: None

Errors: c = 0 means no error, the device driver was opened successfully.

c = 1 means an error condition occurred. The device could not be opened. Possible errors include:

$0020	DRVR_BAD_REQ
$0026	DRVR_NO_RESRC
$0027	DRVR_IO_ERR
$0028	DRVR_NO_DEV
$002F	DRVR_OFF_LINE

An exception is that error code $0024 - DRVR_PRIOR_OPEN indicates that the driver has already been opened. Normal I/O transactions may be requested with no negative effect.

Drvr_Read

Function: This call is used to read data from either a character or block device. A drvr_open call must have been issued to a character device prior to attempting to read data from the device with this call. Block devices do not require (and in fact do not support) a drvr_open call prior to attempting to read data from the device.

Inputs: GS/OS Direct Page

Outputs: GS/OS Direct Page and Buffer contents

Errors: c = 0 means no error, the device driver returned the requested data into the buffer specified on GS/OS direct page.

c = 1 means an error condition occurred, the requested data was not returned. Possible errors include:

$0020	DRVR_BAD_REQ
$0021	DRVR_BAD_PARM
$0023	DRVR_NOT_OPEN (character device only)
$0027	DRVR_IO_ERR
$0028	DRVR_NO_DEV
$002C	DRVR_BAD_COUNT
$002D	DRVR_BAD_BLOCK
$2EXX	DRVR_DISK_SW
$002F	DRVR_OFF_LINE

Drvr_Write

Function: This call is used to write data to either a character or block device. A drvr_open call must have been issued to a character device prior to attempting to write data to the device with this call. Block devices do not require (and in fact do not support) a drvr_open call prior to attempting to write data to the device.

Inputs: GS/OS Direct Page

Outputs: GS/OS Direct Page & Buffer contents

Errors: c = 0 means no error, the requested data was written to the device.

c = 1 means an error condition occurred, data was not written to the device. Possible errors include:

$0020	DRVR_BAD_REQ
$0022	DRVR_BAD_PARM
$0023	DRVR_NOT_OPEN (character device only)
$0027	DRVR_IO_ERR
$0028	DRVR_NO_DEV
$002B	DRVR_WR_PROT
$002C	DRVR_BAD_COUNT
$002D	DRVR_BAD_BLOCK
$2EXX	DRVR_DISK_SW
$002F	DRVR_OFF_LINE

Drvr_Close

Function: This call is used to reset the driver to the pre-open state. This may include releasing of resources such as memory for buffers. Block devices will take n action on this call and should return a 'BADCMD' error. Prior to dispatching to the device, the device dispatcher will check that the DIB for the device specified by the device number indicates that the device is a character device.

Inputs: GS/OS Direct Page

Outputs: None

Errors: c = 0 means no error, the device driver was closed successfully.
c = 1 means an error condition occurred, the driver was not closed. Possible errors include:

$0020	DRVR_BAD_REQ
$0027	DRVR_IO_ERR
$0028	DRVR_NO_DEV
$002F	DRVR_OFF_LINE

Drvr_Status

Function: This call is used obtain specific status information pertaining to a device.

Inputs: GS/OS Direct Page

Outputs: GS/OS Direct Page

Errors: c = 0 means no error, the requested status was returned successfully.
c = 1 means an error condition occurred, no status was returned. Possible errors include:

$0020	DRVR_BAD_REQ
$0021	DRVR_BAD_CODE
$0022	DRVR_BAD_PARM
$0027	DRVR_IO_ERR
$0028	DRVR_NO_DEV
$002B	DRVR_WR_PROT
$2EXX	DRVR_DISK_SW
$002F	DRVR_OFF_LINE

Drvr_Control

Function: This call is used send specific control information or requests to a device.

Inputs: GS/OS Direct Page

Outputs: GS/OS Direct Page

Errors: c = 0 means no error, the control list was sent to the device successfully.
c = 1 means an error condition occurred, no control information was sent to the device. Possible errors include:

$0020	DRVR_BAD_REQ
$0021	DRVR_BAD_CODE
$0022	DRVR_BAD_PARM
$0027	DRVR_IO_ERR
$0028	DRVR_NO_DEV
$002F	DRVR_OFF_LINE

Drvr_Flush

Function: This call is used to output any characters in a character driver's buffer in preparation for purging a driver. Block devices do not support this call and will return with no error. The device driver will check the DIB to make sure the device is a character device prior to dispatching to the device driver.

Inputs: GS/OS Direct Page

Outputs: GS/OS Direct Page

Errors: c = 0 means no error, any device driver maintained buffer was written to the device.

c = 1 means an error condition occurred, any device driver maintained buffer may contain data that was not written to the device. Possible errors include:

$0020	DRVR_BAD_REQ
$0023	DRVR_NOT_OPEN (character device only)
$0027	DRVR_IO_ERR
$0028	DRVR_NO_DEV
$002B	DRVR_WR_PROT
$2EXX	DRVR_DISK_SW
$002F	DRVR_OFF_LINE

CACHE_FIND_BLK $01FC04

Function: This routine will try to find the requested block in the cache. If it's found, it'll be moved to the start of the LRU chain and a 4 byte pointer will be returned to the start of the requested block. The bucket links will not be moved. One of two possible searches may be specified for this call. Driver's cache block by device number while an FST may cache a block by volume ID when a deferred session is in process. A routine calling this system service routine must specify the type of cached block that the search will operate on.

Inputs: GS/OS direct page
Carry Flag = 1 : search for deferred volume (FST's)
Carry Flag = 0 : search for block (Driver's)

Outputs: GS/OS direct page

Errors: c = 0 means no error, the block is in the cache
c = 1 means the block's not in the cache

Notes: Input and output is passed to this routine by GS/OS direct page and full native mode is always assumed.

CACHE_ADD_BLK $01FC08

Function: This routine will try to add the requested block into the cache. It's position within the LRU and bucket chain will be at the start of the list. In the event there is not enough room in the cache, the last recently used block(s) will be purged until there is enough room for the requested block.

Inputs: GS/OS direct page

Outputs: GS/OS direct page

Errors: c = 0 means no error, the block is cached
c = 1 means something screwed up, the block's not cached

Notes: Input and output is passed to this routine by GS/OS direct page and full native mode is always assumed.

CACHE_INIT $01FC0C

Function: This routine will try to initialize the cache. Memory as needed by the cache is obtained from the Mike Memory Manager. 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, unless a shutdown and init sequence occurs.

Inputs: GS/OS direct page

Outputs: GS/OS direct page

Errors: c = 0 means no error, the cache has been initialized
c = 1 means something screwed up, the initialization failed

Notes: Input and output is passed to this routine by GS/OS direct page and full native mode is always assumed.

CACHE_SHUTDN $01FC10

Function: This routine will try to shutdown the cache by deleting each entry on 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.
Inputs: GS/OS direct page
Outputs: GS/OS direct page
Errors: c = 0 means no error, the cache has been shutdown
c = 1 means something screwed up, the cache is unreliable now
Notes: Input and output is passed to this routine by GS/OS direct page and full native
mode is always assumed.

CACHE_DEL_BLK $01FC14

Function: This routine will try to delete the requested block from the cache.
Inputs: GS/OS direct page
Outputs: GS/OS direct page
Errors: c = 0 means no error, the block has been deleted from the cache
c = 1 means something screwed up or the block's not in the cache
Notes: Input and output is passed to this routine by GS/OS direct page and full native
mode is always assumed.

CACHE_DEL_VOL $01FC18

Function: 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 block devices of
the specified FST will be deleted.
Inputs: GS/OS direct page
Outputs: GS/OS direct page
Errors: c = 0 means no error, the device's block(s) have been deleted from the cache
c = 1 means something screwed up, the block(s) may still be in the cache
Notes: Input and output is passed to this routine by GS/OS direct page and full native
mode is always assumed.

Memory Management Call Descriptions

For more detailed specifications of these calls, please the the GS/OS Information Manager ERS.

ALLOC_SEG $01FC1C

alloc_seg(request_size) : vp

Function: This routine returns a Virtual Pointer to a segment of the requested size. If the carry is set upon entry, the segment is filled with zeros (It should be clear if zeroing the segment is unnecessary). If memory could not be obtained the carry is returned set.

RELEASE_SEG $01FC20

release_seg(vp)

Function: This call frees a block of memory obtained with the allocate call pointed to by the virtual pointer passed. If for any reason the memory could not be returned the carry is returned set.

DEREF $01FC38

deref(vp) : pointer

Function: This call returns a pointer corresponding to the current location of the block referenced by the virtual pointer. The 32 bit value returned points to the first useable byte in the block. Don't confuse GIM dereferencing with that associated with the System Memory Manager; they are two different things. A deref gives you a pointer that is good until any system call is made that forces memory to move around (e.g. any VCR/FCR/alloc_seg call). If you wish to ensure that a pointer will not change you need to use the lock_mem and unlock_mem calls.

LOCK_MEM $01FC68

lock_mem()

Function: This call causes all GS/OS managed memory segments created with alloc_seg, alloc_vcr, and alloc_fcr to be locked down with respect to the GS memory manager. It is expected that any routine calling lock_mem will also make a call to unlock_mem; failure to do so could cause the system to run out of memory prematurely.

UNLOCK_MEM $01FC6C

unlock_mem()

Function: All segments created with alloc_seg, alloc_vcr, and alloc_fcr are unlocked.

ALLOC_VCR $01FC24

alloc_vcr(pathname_string, size) : vp

Function: The routine allocates a Volume Control Record, links it into the VCR chain and assigns it a VCR ID (This number is analogous to the FCR's reference number). The size parameter must be at least $000E bytes and may be larger than this if the FST wishes to store it's own info starting at byte $000E.

RELEASE_VCR $01FC28

release_vcr(vcr_id)

Function: This routine deallocates a Volume Control Record and relinquishes the VCR ID.

FIND_VCR $01FC48

find_vcr(vol_name or vcr_id) : vp

Function: A search is done for the VCR whose ID or name are equal to that passed, and a VP is returned corresponding to the first VCR that matches the specified criteria.

RENAME_VCR $01FC58

rename_vcr(vcr_id, pathname)

Function: The name of the VCR referenced is changed to the name specified in the X,Y registers.

ALLOC_FCR $01FC2C

alloc_fcr(size, file_name) : vp

Function: This call allocates a File Control Record and links it into the FCR chain. The reference number is assigned and placed within the FCR.

RELEASE_FCR $01FC30

FIND_FCR $01FC4C

find_fcr(file_name or ref_num) : vp

Function: A search is done for the FCR whose ref_num or name are equal to that passed, and a VP is returned corresponding to the first FCR that matches the specified criteria.

RENAME_FCR $01FC5C

rename_fcr(ref_num, pathname)

Function: The name of the FCR referenced is changed to the name specified in the X,Y registers.

SWAP_OUT $01FC34

swap_out(dev_num)

Function: This routine moves 'offline' any volume in the device specified (A volume is offline if it's 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 specified device which has no open files is deleted from the system.

GET_VCR $01FC60

get_vcr(index) : vp

Function: The system walks the VCR list returning a VP to the nth VCR. By calling this routine with sequential accumulator values, the entire VCR list is walked.

GET_FCR $01FC64

get_fcr(index) : vp

Function: The system walks the FCR list returning a VP to the nth FCR. By calling this routine with sequential accumulator values, the entire FCR list is walked.

GET_SYS_GBUF $01FC3C

Function: This call returns a pointer to a locked, 1Kby segment of memory which should be used by an FST as the general purpose I/O buffer. (The piece of memory allocated for this buffer is selected at boot time to maximize its efficiency as a device driver buffer, and therefore this is the only recommended use for this piece of memory.) The contents of this buffer will not be preserved from call to call since all FSTs share this buffer, though the buffer is guaranteed not to move around from call to call.

Input: None

Output: X <- Pointer to 1K I/O buffer (low)
Y <- Pointer to 1K I/O buffer (high)

Errors: None

SYS_EXIT $01FC40

Function: This is the normal return vector for an FST which was entered through its application entry point via a call from SCM's call_fst routine. SYS_EXIT never returns to the FST. This function processes the error code returned by the FST and restores the D and S registers to the values they had before the FST was called. Thus, the FST may abort processing without "unwinding" the stack.

Input: c = 0, normal return from FST
c = 1, error return from FST

A = error code if c=1.

The processor must be in native mode (e=0).

Output: None

Errors: None

SYS_DEATH $01FC44

Function: Immediately halts execution of GS/OS and calls the System Failure Manager in the Miscellaneous Toolkit to display an error code and the return address of the JSL instruction that called SYS_DEATH.

Input: A --> Error code in low byte. Upper byte = $00.

Currently defined error codes:

$0001	Unclaimed interrupt
$000A	Volume Control Record unusable
$000B	File Control Record unusable
$000C	Block zero allocated illegally
$000D	Interrupt with I/O shadowing off

Exit: It doesn't exit. BANG BANG, your dead!!!

Message: The error message has the following form:

GS/OS System Error

Address = $nnnnnn LC Bank = #n

Error code = $nnnn

Restart

$nnnnnn is the 3 byte return address pushed by the JSL to SYS_DEATH.

$n is the language card bank number (0 or 1) that was switched in when the error occurred.

$nnnn is the error code passed in A.

The user must press RETURN or click on Restart to reboot the system.

SET_SYS_SPEED $01FC50

Input Parameters: A Register contains speed setting as follows:

$0000	= 1 mHz	(Apple//GS Normal Speed)
$0001	= 2.6 mHz	(Apple//GS Fast Speed)
$0002	= >2.6 mHz	(Accelerated Speed)
$0003	= Not Speed Dependent	(Accelerated Speed)

Settings from $0004 through $FFFF are not valid.

Output Parameters: A Register contains speed setting that was in effect prior to issuing this system service call.

This call is intended to provide hardware accelerators with a means of staying compatible with device drivers that may have speed dependent software implementations. The device dispatcher will obtain the device driver's speed class from the DIB and issue a call to this system service call to set the system speed.

An accelerator card may intercept this vector and replace the system service call with it's own routine in order to maintain compatibility with the operating system device drivers.

CACHE_LOCK $01FC54

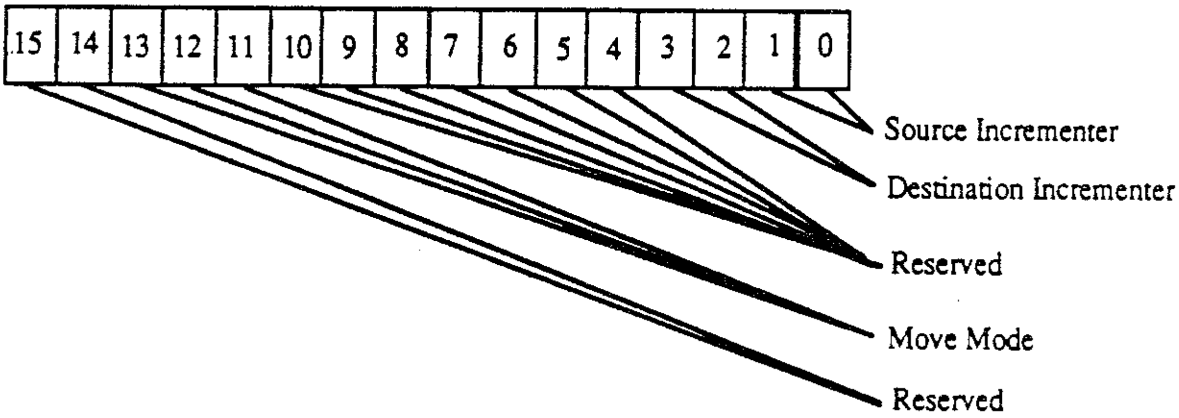
Function: This routine will try to lock or unlock the specified block in the cache. To lock a block, the drvr_enable word should have the hi-bit on. To unlock, the hi-bit
should be 0.
Inputs: GS/OS direct page
Outputs: GS/OS direct page
Errors: c = 0 means no error, the block is locked or unlocked
c = 1 means the block's not in the cache
Notes: Input and output is passed to this routine by GS/OS direct page and full native
mode is always assumed.

MOVE_INFO $01FC70

Function: This call transfers a block of from a source buffer to a destination buffer. 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 banks. The source and/or the destination buffer is allowed to straddle a bank boundary. 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. The high byte of source, dest, and count must be zero. The source and destination blocks must not overlap. This routine executes self modifying code on the stack and therefore is romable and it is reentrant!!! The general move routine in this program is based on the one used by the memory manager. It has been updated for GS/OS purposes.

```
Calling sequence:
  1. Place machine in full native mode (e=0, m=0, x=0)
  2. Push high order word of source pointer onto stack
  3. Push low order word of source pointer
  4. Push high order word of destination pointer
  5. Push low order word of destination pointer
  6. Push high order word of transfer count
  7. Push low order word of transfer count
  8. Push command byte
```

Move Command Byte



```
bits 15/14       = reserved
bits 13/12/11    = move mode
                 = 000          =reserved
                 = 001          =block move
                 = 010-111      =reserved
bits 10/9/8/7    = reserved
bits 6/5/4       = reserved
bits 3/2         = destination incrementer
                 = 00 (+0)      =constant destination
                 = 01 (+1)      =increment destination by 1
                 = 10 (-1)      =decrement destination by 1
                 = 11           =reserved
bits 1/0         = source incrementer
                 = 00 (+0)      =constant source
                 = 01 (+1)      =increment source by 1
```

Predefined command bytes for the move_info routine.

```
moveblkcmd      equ     $0800           ;move block command
```

Most common command

```
move_sinc_dinc  equ     $05+moveblkcmd  ;source incs - destination incs
```

Less common commands

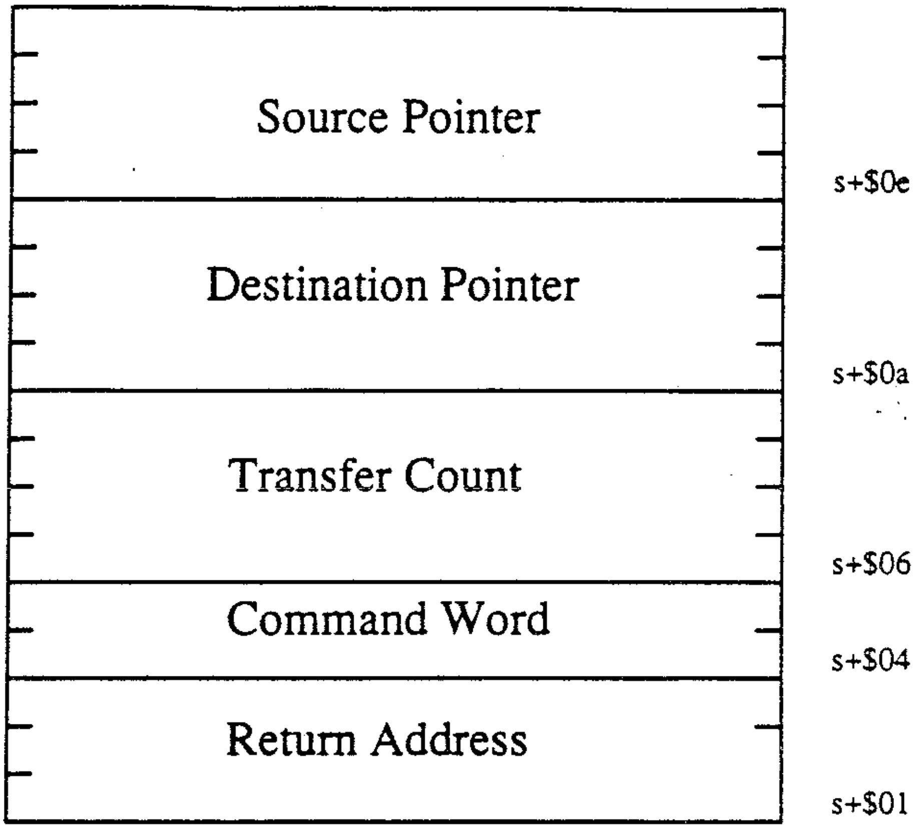
```
move_sinc_ddec  equ     $09+moveblkcmd  ;source incs - destination decs
move_sdec_dinc  equ     $06+moveblkcmd  ;source decs - destination incs
move_sdec_ddec  equ     $0a+moveblkcmd  ;source decs - destination decs

move_scon_dcon  equ     $00+moveblkcmd  ;src constant - destination constant
move_sinc_dcon  equ     $01+moveblkcmd  ;source incs - destination constant
move_sdec_dcon  equ     $02+moveblkcmd  ;source decs - destination constant

move_scon_dinc  equ     $04+moveblkcmd  ;source constant - destination incs
move_scon_ddec  equ     $08+moveblkcmd  ;source constant - destination decs
```

9. jsl Move_Info

Sample code

```
        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
```

Stack on Entry to Move_Info



Outputs:

```
c= 0/1 no error/error
data bank and direct register are preserved
a= error code
x,y= scrambled
```

CVT_0_TO_1 $01FC74

Function: This call converts a class 0 string (a string with a length byte) into a class 1 string (a string with a length word). The source string pointer and destination string pointer are passed as parameters on the stack. The source and destination string areas must either be identical or else completely non-overlapping. However, the routine does not check this.

Inputs:

	Longword	Pointer to source string
	Longword	Pointer to destination string
SP ->		

Outputs: The routine converts the source string and places the result at the location pointed to by the destination string parameter. It squeezes input parameters out of the stack before returning.

Errors: None.

CVT_1_TO_0 $01FC78

Function: This call converts a class 1 string (a string with a length word) into a class 0 string (a string with a length byte). The source string pointer and destination string pointer are passed as parameters on the stack. The source and destination string areas must either be identical or else completely non-overlapping. However, the routine does not check this.

Inputs:

	Longword	Pointer to source string
SP ->	Longword	Pointer to destination string

Outputs: The routine converts the source string and places the result at the location pointed to by the destination string parameter. It squeezes input parameters out of the stack before returning.

Errors:

c=0 Conversion successful.

c=1 Input string was too long to convert (i.e. more than 255 characters). In this case, the input string is unchanged.

REPLACE_80 $01FC7C

Function: This call replaces all of the colons (ASCII $3A) in a class 1 input string with a character specified by the caller. Typically, this routine is used to convert the internal representation of a pathname to an external representation in which "/" represents the separator. If the input string contains an occurrence of the specified replacement character, this routine returns an error and leaves the input string as is.

Inputs:

	Longword	Pointer to input string
	Word	Replacement character in low byte
SP ->		

Outputs: The input string is converted in place. The routine squeezes input parameters out of the stack before returning.

Errors:

c=0 Conversion successful.

c=1 Input string contained occurrences of the replacement character. In this case, the input string is unchanged.

RESERVED_01 $01FC80

Input: Unspecified

Output: Unspecified

Function: **This system service call is reserved for use by Apple Computer, Inc. This call provides a temporary function. Future versions of GS/OS most likely will not support the function provided by this call.**

RESERVED_02 $01FC84

Input: Unspecified

Output: Unspecified

Function: **This system service call is reserved for use by Apple Computer, Inc. This call provides a temporary function. Future versions of GS/OS most likely will not support the function provided by this call.**

SIGNAL $01FC88

Function: This call signals the occurrence of a specific event and specifies the machine environment parameters to be used when executing the event handler for the event.

Remainder of call description to be supplied when firm. See *Interrupt and Event Management in GS/OS.*

RESERVED_03 $01FC8C

Input: Unspecified

Output: Unspecified

Function: This system service call is reserved for use by Apple Computer, Inc. This call provides a temporary function. Future versions of GS/OS most likely will not support the function provided by this call.

SET_DISKSW $01FC90

Input: The device number present on GS/OS direct page specifies which device will have it's dispatcher maintained disk switched status set.

A Register:	Unspecified
X Register:	Unspecified
Y Register:	Unspecified
Data Bank Register:	Unspecified
Direct Page Register:	Unspecified
P Register:	0=e=m=x

The ROM must not be switched in during this call.

Output:

A Register:	Unspecified
X Register:	Unspecified
Y Register:	Unspecified
Data Bank Register:	Unchanged
Direct Page Register:	Unchanged
P Register:	0=e=m=x

Function: This system service call sets the device dispatcher maintained disk switched error for the device specified by the value in the accumulator. This call supports device drivers that implement device specific status calls which may detect either an OFFLINE or DISKSWITCH condition. These conditions are returned as a status rather than an error and will not be detected by the device dispatcher on exit from the driver call. It is neccessary for the driver to specifically request that the disk switched status be set in this situation. The SET_DISKSW call will in turn call both SWAP_OUT and DEL_CACHE_VOL if the device dispatcher maintained disk switched error was not previously set.

NOTE: If the current device is a linked device, then SET_DISKSW call will in turn call both SWAP_OUT and DEL_CACHE_VOL for each of the linked devices starting with the head link device and proceeding through each forward linked device until reaching the end of the forward linked list. Disk switch maintenance is only performed for a device if the device dispatcher maintained disk switched error for that device was not previously set.

RESERVED_04 $01FC94

This system service call is reserved for use by Apple Computer, Inc.

RESERVED_05 $01FC98

This system service call is reserved for use by Apple Computer, Inc.

RESERVED_06 $01FC9C

This system service call is reserved for use by Apple Computer, Inc.

RESERVED_07 $01FCA0

This system service call is reserved for use by Apple Computer, Inc.

SUP_DRVR_DISP $01FCA4

This system service call is the main entry point in the supervisory driver dispatcher. Supervisory drivers provide an interface for higher level device drivers to access hardware. Supervisory driver calls can be classified into one of two groups. Calls with a supervisory driver number of zero are calls to the supervisory dispatcher and will not be passed on to a supervisory driver. Calls with a supervisory driver number of nonzero will be passed on to the supervisory driver specified by the supervisory driver number.

The following calls must are supported by the supervisory dispatcher and will not be passed on to a supervisory driver.

Driver Number	Call Number	Function
$0000	$0000	Return driver number for ID 'n'
$0000	$0001	Set SIB pointer
$0000	$0002 - $FFFF	Reserved

Get Supervisor Driver Number

Call Input Parameters:	A Reg:	Supervisor Driver Number	= $0000
	X Reg:	Supervisor Call Number	= $0000
	Y Reg:	Supervisor ID Number	= $xxxx
	DirectPage:	SIB Pointer	
Call Output Parameters:	A Reg:	Error Code	
	X Reg:	Supervisory driver number	

Supervisor Call Number: This word parameter specifies which type of call is to be issued to the supervisory driver.

Supervisor Driver Number: This word parameter is returned as output from this call and indicates the device number of the supervisory driver indicated by the supervisor ID number passed as an input.

SIB Pointer: This longword points to the supervisor information block for the supervisory driver being accessed. This parameter is set up by the supervisory driver dispatcher.

This call is issued by a device driver to determine what supervisory driver number should be used in calling the supervisory driver associated with that device driver. This call is handled by the supervisory device dispatcher and does not result in any execution of a supervisory driver. The device driver passes to the supervisor driver dispatcher during it's startup call, the supervisor ID number for the supervisory driver that it wishes to use. The supervisor driver dispatcher will return the driver number that indicates the supervisory drivers position in the supervisory driver list. This number is passed by the device driver to the supervisory driver dispatcher on all subsequent calls to the supervisory driver. Note that if the supervisory driver dispatcher cannot find a supervisor driver for the supervisory driver ID number passed by the device driver than an error "no device" will be returned. In this case the device driver will not be able to use the supervisory driver and should return an error during it's startup call.

Set SIB Pointer

Call Input Parameters:	A Reg:	Supervisor Driver Number	= $0000
	X Reg:	Supervisor Call Number	= $0001
	Y Reg:	Supervisor Number to set SIB pointer	
	DirectPage:	SIB Pointer	
Call Output Parameters:	A Reg:	Error Code	
	X Reg:	Supervisory driver number	

Supervisor Call Number: This word parameter specifies which type of call is to be issued to the supervisory driver.

Supervisor Driver Number: This word parameter is returned as output from this call and indicates the device number of the supervisory driver indicated by the supervisor ID number passed as an input.

SIB Pointer: This longword points to the supervisor information block for the supervisory driver being accessed. This parameter is set up by the supervisory driver dispatcher.

This call is may be issued to set the SIB pointer on GS/OS direct page to the SIB specified by the supervisory driver number passed as input in the Y register.

SUPERVISOR STARTUP

Call Input Parameters:	A Reg:	Supervisor Driver Number	≠ $0000
	X Reg:	Supervisor Call Number	= $0000
	DirectPage:	SIB Pointer	
Call Output Parameters:	A Reg:	Error Code	

Supervisor Call Number: This word parameter specifies which type of call is to be issued to the supervisory driver.

Supervisor Driver Number: This word parameter specifies which supervisory driver is to be started.

SIB Pointer: This longword points to the supervisor information block for the supervisory driver being accessed. This parameter is set up by the supervisory driver dispatcher.

This call is responsible to prepare the supervisory driver for use by device drivers. Any system resources required by the supervisory driver, such as memory, should be allocated during this call. If the supervisor cannot allocate resources neccessary to support and device driver calls then the supervisory driver should return an error. If a supervisory driver returns an error as a result of the startup call then the supervisory driver will be purged from the supervisory driver list.

SUPERVISOR SHUTDOWN

Call Input Parameters:	A Reg:	Supervisor Driver Number	≠ $0000
	X Reg:	Supervisor Call Number	= $0001
	DirectPage:	SIB Pointer	
Call Output Parameters:	A Reg:	Error Code	

Supervisor Call Number: This word parameter specifies which type of call is to be issued to the supervisory driver.

Supervisor Driver Number: This word parameter specifies which supervisory driver is to be started.

SIB Pointer: This longword points to the supervisor information block for the supervisory driver being accessed. This parameter is set up by the supervisory driver dispatcher.

This call is responsible for releasing any system resources acquired during startup of the supervisory driver. Supervisory drivers are shutdown after all device drivers have been shutdown.

Driver/Supervisor Specific Calls

Call Input Parameters: $FFFF	A Reg: X Reg:	Supervisor Driver Number Supervisor Call Number	≠ $0000 = $0002
	DirectPage:	SIB Pointer	
Call Output Parameters:	A Reg:	Error Code	

Supervisor Call Number: This word parameter specifies which type of call is to be issued to the supervisory driver.

Supervisor Driver Number: This word parameter is returned as output from this call and indicates the device number of the supervisory driver indicated by the supervisor ID number passed as an input.

SIB Pointer: This longword points to the supervisor information block for the supervisory driver being accessed. This parameter is set up by the supervisory driver dispatcher.

These calls are used by device drivers to request specific tasks to be completed by the supervisory driver. These calls are designed to accommodate the needs of all device drivers using a specific supervisory driver.

INSTALL_DRIVER $01FCA8

In order to support removable partitionable media on block devices it is neccessary to be able to dynamically install devices in the device list as new partitions come online. A call 'INSTALL_DRIVER' has been provided for this purpose. Note that this call implies that the GS/OS device list can grow. There is no mechanism that can cause devices to be removed from the device list.

Call Input Parameters:	X Reg:	DIB Address (low word)
	Y Reg:	DIB Address (high word)
Call Output Parameters:	A Reg:	Error Code

DIB Address: This longword specifies the address of a list of device information blocks to be installed into the device list. The first longword in this list specifies the number of device information blocks to be installed. This is followed by a longword pointer to a device information block for each DIB to be installed.

This call is used to dynamically install a list of drivers into the device list. This call does not install the driver immediately, rather it informs the device dispatcher that the drivers are to be installed. The device dispatcher will attempt to install the drivers at the end of the current device call if a device call is in progress or at the end of the next device call if no device call is in progress. When attempting to install drivers via this system service call error parsing is absolutely required. Two possible errors may be returned from this call. If an out of memory error is returned then it will not be possible to install any drivers. If however a driver busy error is returned then a post driver install is already pending. In this case the driver installation call must be deferred until then next access to the device driver which is installing additional devices.

When installing the driver, the device dispatcher will insert the device into the device list and issue 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 then the device will not be installed into the device list.

Note that there is no indication to an application that the device list has changed size. Applications that scan block devices (such as the Finder) should issue a D_INFO call with a pcount of $0003 to get the device's characteristics. If the device is a block device with removable media then a status call should be issued to the device. This scan operation should always begin with device $0001 and continue up the device list until a device not found error occurs. If applications scan devices in this manner dynamically installed devices will always be included in the scan operation.

GET_BOOT_PFX $01FCAC

This call returns a pointer to the boot volume name.

Call Input Parameters: none

Call Output Parameters: X Reg: Low part of pointer to GString of boot volume name
Y Reg: High part of pointer to GString of boot volume name

No errors can be returned on this call.

SET_BOOT_PFX $01FCB0

This call allows the changing of the boot prefix. GQuit is updated with the new information provided by this call.

Call Input Parameters: X Reg: Low part of pointer to GString of boot volume name
Y Reg: High part of pointer to GString of boot volume name

Call Output Parameters: none

Memory manager errors can be returned on this call if memory is unavailable.

DYN_SLOT_ARBITER $01FCBC

In order to support both internal and external slots in the future, a mechanism must be in place to dynamically select an internal or external slot. This task must include swapping in any screen hole memory associated with any given slot. The slot arbiter provides this mechanism by maintaining an image if each internal and external slot's screen hole memory locations (including slot zero screen holes associated with any given slot).

Call Input Parameters:	A Reg:	Requested slot
	X Reg:	Undefined
	Y Reg:	Undefined
Call Output Parameters:	Carry Flag:	Cleared if requested slot was granted. Set if requested slot was denied.

Requested Slot: This word specifies the slot to be requested where bits 0 through 2 indicate the slot number and bit 3 indicates that the requested slot is external or internal. Bit 3 will be set for external slots. All other bits in the requested slot must be zero.

NOTE: No dynamic slot arbitration has been implemented at this time. The current implementation merely checks if the requested slot has been selected by examining SLTROMSEL or RDC3ROM. If the requested slot is not currently selected then the request will be denied. A full implementation of the slot arbiter can be expected at a future date. Hang in there!!!

$01FCD8 Unbind_Int_Vector