To: Cortland Software Group
Software ERS binder

From: Rich Williams

Subject: **Memory Managers are a girl's best friend**

or

If Napoleon had a memory manager we would all be speaking French today

---

Revision history

Rev 1	Nov. 5, 1985	First pass
Rev 2	Nov. 27, 1985	PurgeAll, Lockall, etc. added Properties of blocks added.
Rev 3	Feb. 10, 1986	Copy commands added. Call numbers added. The order of parameters changed. Error codes added. TotalMem call added
Rev 4	Mar. 10,1986	Standard calls added. **Parameters added to AppInit and AppQuit.** SetPurgeAll parameters switched.

Table of Contents

1.0 Introduction

The memory manager is responsible for allocating blocks of memory to programs. It does the overall bookkeeping of what memory is being used ands keeps track of who owns various blocks of memory.

2.0 Design Philosophy

Up until now, memory manager discussions have been about a SOS-like memory manager that allocates memory blocks in pages and does not perform relocation or compaction of memory.

The SOS memory manager allocates memory in pages. This works well with a program that has a single large data area that grows. An example of this might be a word processor that has a document that grows as the user types it in. Maintaining a single large data block will be difficult on a 256K machine because of the fragmentation of free memory by video screens, I/O space, etc. The SOS memory manager also does not relocate memory blocks. Consequently, you can only grow a block if the adjacent space is free. This forces programs to allocate more space than they need to insure that they can grow if they need to. If a program wants to have more than one block that grows, it must carefully allocate the blocks far apart from each other. In short, the programs end up doing almost all of their own memory management!

Both AppleWorks and Catalyst are programs that wrote their own Maclike memory managers. Since these are the types applications that we are most interested in supporting, we should give them the memory tools they require.

There are disadvantages to this scheme. It will take longer to write. It will take a lot more room in the rom. And, in making a more powerful memory manager, it could become too complicated or slow for people to use.

3.0 Differences from the Mac memory manager

There are many differences between this memory manager and the Macintosh's. Some of the most notable are:

Fewer calls. If you want others, ask for them
Blocks now have an owner ID
There is now a purging priority level for the block
There are only handles and no pointers

4.0 Properties of memory blocks

Memory blocks have attributes that determine how they are allocated and maintained. Some attributes are defined at allocation time and can't be changed. Others can be modified after allocation.

4.1 Allocation attributes

The memory in the Apple // and the architecture of the 65816 force many restrictions on how blocks can be allocated. Blocks, for example, may have to be page aligned or they may have to be in a certain bank. When allocating a block, an attributes word is specified that determines how the block is allocated. These attributes can only be set when the block is allocated. The attributes are:

D14	Fixed
D4	May not cross bank boundary
D3	May not use special memory
D2	Page Aligned
D1	Fixed Address
D0	Fixed Bank

Dn = bit in attributes word. D15 = msb D0 = lsb

Fixed:
If a block is fixed, it cannot be moved when compacting memory. Code blocks will usually be fixed, but data blocks should usually not be.

May not cross a bank boundary:
This specifies that a block must not cross banks. Code blocks, for example, may never cross banks.

May not use special memory:
This specifies that the block may not be allocated in special memory. This is memory that is used in the Apple //e and includes banks 0, 1 and the video screens.

Page aligned:
For timing reasons, code or data may need to be page aligned

Fixed Address:
This is used to specify that the block must be at a specified address when allocated. An example is allocating the graphics screen.

Fixed Bank:
This specifies that the block must start in a specified bank. An example is allocating a block to be used as a zero page or stack.

4.2 Modifiable attributes

The memory manager can move or purge a block while making room for a new block. There are attributes that determine whether a block can be moved or purged. These attributes can be changed by the user after a block is created. The attributes are:

D15 Locked
D9.8 PurgeLevel

Locked:
When a block is locked, it is unmovable and unpurgeable irregardless of what Movable or PurgeLevel is set to. This feature is to allow a block to be temporily locked down while it is being executed or referenced.

PurgeLevel:
This is a two bit number defining the purging priority of a block. 0 is unpurgable and level 3 is the first purged.

5.0 Memory Manager Calls

Calls to the memory manager fall into the following catagories:
* = call not implemented in alpha 2.0 rom

Standard calls		
$01	MMBootInit	Boot time initialization
$02	MMAppInit	Application initialization
$03	MMAppQuit	Application quit call
$04	MMGetVersion	Gets version number
$05	MMReset	Called by system reset
$06	MMStatus	Active status
Allocating memory		
$09	NewHandle	Creates a new block and handle
* $0A	ReallocHandle	Uses an existing handle
Freeing memory		
$10	DisposHandle	Deallocates a handle
$11	DisposAll	Deallocates all of an owner's memory
$12	PurgeHandle	Purges the contents of a block
$13	PurgeAll	Purges all of an owner's purgable blocks
Information on blocks and memory		
$18	GetHandleSize	Gets the size of a block
$19	SetHandleSize	Grows or shrinks a block
$1A	FindHandle	Finds the handle of a block containing an address
$1B	FreeMem	Gets total amount of free space
* $1C	MaxBlock	Gets size of largest free block
$1D	TotalMem	Gets size of all memory
Other properties of blocks		
$20	HLock	Locks a block
$21	HLockAll	Locks all of an owners blocks
$22	HUnlock	Unlocks a block
$23	HunlockAll	Unlocks all of an owner's blocks
$24	SetPurge	Sets the purge level of a block
$25	SetPurgeAll	Sets the purge level of all of an owner's blocks

	Copying Data		
*	$28	PtrtoHand	Copies from an address to a handle
*	$29	HandtoPtr	Copies from a handle to an address
*	$2A	HandtoHand	Copies from one handle to another
	$2B	BlockMove	Copies from one address to another

5.1 Data types

These are the data types used in the calls:

```
Pointer     = ^Byte
Handle      = ^Pointer
UserID      = Word      {Identifies the owner of a segemnt}
Address     = Long int  {4 byte address}
Size        = Long int  {4 byte size of a block}
PurgeLevel  = 0..3      {Priority to purge a block}
```

5.2 Standard calls

These are standard calls defined for every tool. Note that the AppInit call is different from other tool sets.

5.2.1 MMBootInit

This call initializes the memory manager at boot time. An application must never make this call since it will destroy all currently allocated blocks including the caller. Never, ever, ever make this call. Don't even try to use it as part of a protection scheme.

5.2.2 MMAppInit

inputs: none

output: Owner: UserID

This call is made by an application when it starts up. If the call is not made from a valid segment, a IDErr is returned. If this happens, the program should call the ID Manager for a ID number and then call the memmory manager to allocate its program segments. This should only happen when running under the current operating systems.

5.2.3 MMAppQuit

inputs: Owner: UserID

output: none

This call is given to the memory manager by the application when it is finished and is about to exit.

5.2.4 MMGetVersion

inputs: none

output: Version: word

This returns the version number of the memory manager.

5.2.5 MMReset

This is an internal call used by the system at reset time. An application should never make this call.

5.2.6 MMStatus

inputs: none

outputs: Status: Boolean (always true)

Status is used to test if the tool is active. The memory manager is always active.

5.3 Allocating memory

These commands are used to create memory blocks.

5.3.1 NewHandle

inputs:	BlockSize:	Size
	Owner:	UserID
	Attributes:	Word
	Location:	Address
outputs:	Handle	

NewHandle is used to create a new block. BlockSize is the size of the block to create. The attributes are described in section 4. If a block of size 0 is created, the handle will be set to NIL.

5.3.2 ReallocHandle

inputs:	TheHandle:	Handle
	BlockSize:	Size
	Owner:	UserID
	Attributes:	Word
	Location:	Address
output:	none	

ReallocHandle is used to reallocate a block that has been purged. BlockSize is the size of the block to create. The attributes are described in section 4 Any information that was in the purged block has been lost.

5.4 Freeing memory

These commands are used to free blocks and pointers. Once a block or handle is freed, its contents cannot be recovered.

5.4.1 DisposHandle

inputs:	theHandle:	Handle
output:	none	

DisposHandle purges the block specified by theHandle and deallocates the handle. The block is purged irregardless of its purge level but it must

be unlocked.

5.4.2 DisposAll

inputs: Owner: UserID

output: none

DisposAll disposes all of the handles owned by Owner.

5.4.3 PurgeHandle

inputs: theHandle: Handle

output: none

EmptyHandle purges the block specified by theHandle. The block is purged irregardless of its purge level but it must be unlocked. The handle itself remains allocated but is pointed to NIL.

5.4.4 PurgeAll

inputs: Owner: UserID

output: none

PurgeAll purges all of the purgable blocks owned by Owner

5.5 Information on Blocks

These commands are used to grow or shrink memory blocks.

5.5.1 GetHandleSize

inputs: theHandle: Handle

output: Size

GetHandleSize returns the size of a block specified by theHandle.

5.5.2 SetHandleSize

inputs: newSize: Size
theHandle: Handle

output: none

SetHandleSize changes the size of the block specified by theHandle. The block can be made larger or smaller. If necessary to lengthen a block, memory may be compacted or blocks may be purged. The handle should be unlocked since it may have to move to change size. If the size is set to 0, the handle will be set to NIL.

5.5.3 FindHandle

inputs: Location: Address

output: theHandle: Handle

FindHandle returns the handle to the block containg the address specified by location. Note that if the block is not locked, it may move. If the address is not in any handle, then NIL (0) is returned.

5.5.4 FreeMem

inputs: none

output: Size

FreeMem returns the total number of free bytes in memory. It does not count memory that could be freed by purging. Because of memory fragmentation, it may not be possible to allocate a block this large. FreeMem does a compaction of the memory space.

5.5.5 MaxBlock

inputs: none

output: size

MaxBlock returns the size of the largest free block in memory. It

does not count memory that could be freed by purging or compacting.

5.5.6 TotalMem

inputs: none

output: size

TotalMem returns the size off all of the memory in the machine. This includes the main 256K.

5.6 Other properties of blocks

These commands change the other properties of memory blocks.

5.6.1 HLock

inputs: theHandle: Handle

output: none

HLock locks a block specified by theHandle. A locked block cannot be relocated or purged during memory compaction.

5.6.2 HLockAll

inputs: Owner: UserID

output: none

HLockAll locks all of the blocks owned by Owner.

5.6.3 HUnlock

inputs: theHandle: Handle

output: none

HUnlock unlocks a block specified by theHandle. A unlocked block can be relocated during memory compaction.

5.6.4 HUnlockAll

inputs: Owner: UserID

output: none

HUnlockAll unlocks all of the blocks owned by Owner.

5.6.5 SetPurge

inputs: newPlevel: PurgeLevel
theHandle: Handle

output: none

SetPurge sets the PurgeLevel of the block specified by theHandle to newPlevel.

5.6.6 SetPurgeAll

inputs: newPlevel: PurgeLevel
Owner: UserID

output: none

SetPurgeAll sets the purge level of all of the blocks owned by Owner.

5.7 Copying Data

These commands are used to copy data from one place to another in the machine.

5.7.1 PtrtoHand

5.7.2 HandtoPtr

5.7.3 HandtoHand

Comming soon to a memory manager near you.

5.7.4 BlockMove

inputs:	Source:	Pointer
	Dest:	Pointer
	Count:	Size
output:	none	

BlockMove copies Count bytes from Source to Dest. BlockMove works correctly even if the source and destination blocks overlap or cross bank boundaries. Beware that no address validation is done and BlockMove will cheerfully write over anything when told to do so.

6.0 Error Codes

These are the error codes returned by the memory manager

Code	Type of error	
$0000	No error	
$0201	MFullErr	Memory full error
$0202	NilErr	Illegal operation on a Nil handle
$0203	NotNilErr	A Nil handle was expected for this operation
$0204	LockErr	Illegal operation on a locked or immovable block
$0205	PurgeErr	Attempt to purge an unpurgable block
$0206	HandleErr	An invalid handle was given
$0207	IDErr	An invalid owner ID was given