Memory Manager Reference

Carbon > Resource Management

2007-06-27

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Simultaneously published in the United States and Canada.

Contents

Memory Manager Reference

Framework: CoreServices/CoreServices.h

Declared in MacMemory.h

Overview

The Memory Manager was the memory management solution for versions of the Macintosh operating system prior to Mac OS X. It remains available for compatibility with legacy applications and for new applications that must work with legacy Carbon code that requires handles.

In previous versions of the Macintosh operating system, developers used the Memory Manager to set up an application's memory partition at launch time, to manage an application's heap, to minimize application memory fragmentation, and to implement a scheme to avoid low-memory conditions. All of these operations are now handled transparently by Mac OS X.

In the Mac OS X Memory Manager, many functions are deprecated, the callbacks are not functional, and the data types and constants are not used. In Mac OS X there is no need to set up or manage an application memory partition or to manage pointers. To allocate memory, in most cases you simply use the C functions `malloc` or `calloc`. Mac OS X ensures that every application has access to as much memory as it needs—up to 4 gigabytes of addressable space per 32-bit process.

Mac OS X does not support functions for accessing the system heap, as the system heap is unavailable to applications in Mac OS X. Starting with Mac OS X v10.3, Memory Manager is thread safe, and the `MemError` (page 27) function now returns error codes on a per-thread basis.

For information on memory management issues when porting a legacy Macintosh application to Mac OS X, refer to the Carbon Porting Guide and to Technical Note 2130, Memory Allocation Recommendations on Mac OS X.

Functions by Task

Allocating and Releasing Nonrelocatable Blocks of Memory

`DisposePtr` (page 17)
Releases memory occupied by a nonrelocatable block.

`NewPtr` (page 30)
Allocates a nonrelocatable block of memory of a specified size.

`NewPtrClear` (page 31)
Allocates a nonrelocatable block of memory of a specified size with all its bytes set to 0.

Allocating and Releasing Relocatable Blocks of Memory

`DisposeHandle` (page 16)
Releases memory occupied by a relocatable block.

`NewEmptyHandle` (page 28)
Initializes a new handle without allocating any memory for it to control.

`NewHandle` (page 28)
Allocates a new relocatable memory block of a specified size in the current heap zone.

`NewHandleClear` (page 29)
Allocates a relocatable block of memory of a specified size with all its bytes set to 0.

Allocating Temporary Memory

`TempNewHandle` (page 36)
Allocates a new relocatable block of temporary memory.

Assessing Memory Conditions

`MemError` (page 27)
Determines if an application's last direct call to a Memory Manager function executed successfully.

`LMGetMemErr` (page 26)
Returns the result of the last Memory Manager function without clearing the value.

`LMSetMemErr` (page 27)
Sets the value which will be returned by the `MemError` function.

`MaxBlock` (page 70) Deprecated in Mac OS X v10.5
Returns a fixed value for block size that is compatible with most applications. (Deprecated. There is no replacement function; you can assume that any reasonable memory allocation will succeed.)

`StackSpace` (page 71) Deprecated in Mac OS X v10.5
Returns the amount of space between the bottom of the stack and the top of the application heap. (Deprecated. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

Changing the Sizes of Relocatable and Nonrelocatable Blocks

`GetHandleSize` (page 19)
Returns the logical size of the relocatable block corresponding to a handle.

`GetPtrSize` (page 19)
Returns the logical size of the nonrelocatable block corresponding to a pointer.

`SetHandleSize` (page 35)
Changes the logical size of the relocatable block corresponding to the specified handle.

`SetPtrSize` (page 35)
Changes the logical size of the nonrelocatable block corresponding to a pointer.

Managing Relocatable Blocks

`EmptyHandle` (page 18)
Purges a relocatable block and sets the corresponding handle's master pointer to `NULL`.

`HLockHi` (page 23)
Sets the lock bit on the block.

`ReallocateHandle` (page 33)
Allocates a new relocatable block of a specified size and sets a handle's master pointer to point to the new block.

`RecoverHandle` (page 34)
Returns a handle to a relocatable block pointed to by a specified `pointer`.

Manipulating Blocks of Memory

`BlockMove` (page 13)
Copies a sequence of bytes from one location in memory to another.

`BlockMoveData` (page 14)

`BlockMoveDataUncached` (page 15)

`BlockMoveUncached` (page 15)

`BlockZero` (page 16)

`BlockZeroUncached` (page 16)

`HandAndHand` (page 20)
Concatenates two relocatable blocks.

`HandToHand` (page 20)
Copies all of the data from one relocatable block to a new relocatable block.

`PtrAndHand` (page 31)
Concatenates part or all of a memory block to the end of a relocatable block.

`PtrToHand` (page 32)
Copies data referenced by a pointer to a new relocatable block.

`PtrToXHand` (page 33)
Copies data referenced by a pointer to an existing relocatable block.

Setting the Properties of Relocatable Blocks

`HClrRBit` (page 21)
Clears the resource flag of a relocatable block.

`HGetState` (page 22)
Returns a signed byte representing the current properties of a relocatable block.

`HLock` (page 22)
Prevents a relocatable block from moving within its heap zone.

`HSetRBit` (page 23)
Sets the resource flag of a relocatable block.

`HSetState` (page 24)
Restores the properties of a relocatable block.

`HUnlock` (page 25)
Allows a relocatable block to move in its heap zone.

Miscellaneous

`IsHandleValid` (page 25)
Checks that a handle is valid.

`IsHeapValid` (page 26)
Always returns true in Mac OS X.

`IsPointerValid` (page 26)
Checks that a pointer is valid.

Deprecated Functions

You should avoid using the functions listed in this section.

`FreeMem` (page 69) Deprecated in Mac OS X v10.5
Returns the total amount of free space in the current heap zone. (Deprecated. There is no replacement function; you can assume that any reasonable memory allocation will succeed.)

`MaxMem` (page 71) Deprecated in Mac OS X v10.5
Returns the size, in bytes, of the largest contiguous free block in the current heap zone. (Deprecated. There is no replacement function; you can assume that any reasonable memory allocation will succeed.)

`TempDisposeHandle` (page 72) Deprecated in Mac OS X v10.5
Releases a relocatable block in the temporary heap. (Deprecated. Use `DisposeHandle` (page 16) instead; Mac OS X does not have a separate temporary memory heap.)

`CheckAllHeaps` (page 49) Deprecated in Mac OS X v10.4
Checks all known heaps for validity. (Deprecated. There is no replacement function; an application has access only to its own heap in Mac OS X.)

`CompactMem` (page 49) Deprecated in Mac OS X v10.4
Compacts the heap by moving relocatable blocks as needed. (Deprecated. There is no replacement function; memory compaction is never needed and never performed in Mac OS X.)

`DisposeGrowZoneUPP` (page 50) Deprecated in Mac OS X v10.4
(Deprecated. There is no replacement function; heaps never grow in Mac OS X, so the grow-zone function is never called.)

`DisposePurgeUPP` (page 50) Deprecated in Mac OS X v10.4
(Deprecated. There is no replacement function; heaps are never purged in Mac OS X, so the purge function is never called.)

`DisposeUserFnUPP` (page 51) Deprecated in Mac OS X v10.4
(Deprecated. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

`FlushMemory` (page 51) Deprecated in Mac OS X v10.4

Makes a portion of the address space clean. (Deprecated. There is no replacement; this function does nothing in Mac OS X.)

`GetGrowZone` (page 51) Deprecated in Mac OS X v10.4

Returns the current heap zone's grow-zone function. (Deprecated. There is no replacement function; heaps never grow in Mac OS X, so the grow-zone function is never used.)

`GZSaveHnd` (page 52) Deprecated in Mac OS X v10.4

Returns a relocatable block to be protected during grow-zone operations. (Deprecated. There is no replacement; this function does nothing in Mac OS X.)

`HNoPurge` (page 52) Deprecated in Mac OS X v10.4

Marks a relocatable block as unpurgeable. (Deprecated. There is no replacement function; heaps are never purged in Mac OS X.)

`HoldMemory` (page 53) Deprecated in Mac OS X v10.4

Makes a portion of the address space resident in physical memory and ineligible for paging. (Deprecated. There is no replacement; this function does nothing in Mac OS X.)

`HPurge` (page 54) Deprecated in Mac OS X v10.4

Marks a relocatable block as purgeable. (Deprecated. There is no replacement function; heaps are never purged in Mac OS X.)

`InvokeGrowZoneUPP` (page 54) Deprecated in Mac OS X v10.4

(Deprecated. There is no replacement function; heaps never grow in Mac OS X, so the grow-zone function is never called.)

`InvokePurgeUPP` (page 55) Deprecated in Mac OS X v10.4

(Deprecated. There is no replacement function; heaps are never purged in Mac OS X, so the purge function is never called.)

`InvokeUserFnUPP` (page 55) Deprecated in Mac OS X v10.4

(Deprecated. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

`LMGetApplZone` (page 56) Deprecated in Mac OS X v10.4

(Deprecated. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

`LMGetSysZone` (page 56) Deprecated in Mac OS X v10.4

(Deprecated. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

`LMSetApplZone` (page 57) Deprecated in Mac OS X v10.4

(Deprecated. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

`LMSetSysZone` (page 57) Deprecated in Mac OS X v10.4

(Deprecated. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

`MakeMemoryNonResident` (page 57) Deprecated in Mac OS X v10.4

Makes pages in the specified range immediately available for reuse. (Deprecated. There is no replacement; this function does nothing in Mac OS X.)

`MakeMemoryResident` (page 58) Deprecated in Mac OS X v10.4

Makes a portion of the address space resident in physical memory. (Deprecated. There is no replacement; this function does nothing in Mac OS X.)

`MoreMasterPointers` (page 58) Deprecated in Mac OS X v10.4

Allocates a specified number of master pointers in the current heap zone. (Deprecated. There is no replacement function; master pointers do not need to be pre-allocated in Mac OS X.)

`MoreMasters` (page 59) Deprecated in Mac OS X v10.4

Allocates a block of master pointers in the current heap zone. (Deprecated. There is no replacement function; master pointers do not need to be pre-allocated in Mac OS X.)

`MoveHHi` (page 60) Deprecated in Mac OS X v10.4

Moves a relocatable block as high in memory as possible. (Deprecated. There is no replacement function; there is no benefit to moving handles high in memory in Mac OS X.)

`NewGrowZoneUPP` (page 61) Deprecated in Mac OS X v10.4

(Deprecated. There is no replacement function; heaps never grow in Mac OS X, so the grow-zone function is never called.)

`NewPurgeUPP` (page 61) Deprecated in Mac OS X v10.4

(Deprecated. There is no replacement function; heaps are never purged in Mac OS X, so the purge function is never called.)

`NewUserFnUPP` (page 61) Deprecated in Mac OS X v10.4

(Deprecated. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

`PurgeMem` (page 62) Deprecated in Mac OS X v10.4

Purges the current heap zone until the specified number of bytes are available. (Deprecated. There is no replacement; heaps are never purged in Mac OS X, so this function does nothing.)

`PurgeSpace` (page 63) Deprecated in Mac OS X v10.4

Determines the total amount of free memory and the size of the largest allocatable block in the current heap zone if it were purged. (Deprecated. There is no replacement; heaps are never purged in Mac OS X.)

`PurgeSpaceContiguous` (page 63) Deprecated in Mac OS X v10.4

(Deprecated. There is no replacement; heaps are never purged in Mac OS X.)

`PurgeSpaceTotal` (page 63) Deprecated in Mac OS X v10.4

(Deprecated. There is no replacement; heaps are never purged in Mac OS X.)

`ReleaseMemoryData` (page 64) Deprecated in Mac OS X v10.4

Releases the data of a portion of the address space. (Deprecated. There is no replacement; this function does nothing in Mac OS X.)

`ReserveMem` (page 64) Deprecated in Mac OS X v10.4

Reserves space for a block of memory as close to the bottom of the current heap zone as possible. (Deprecated. There is no replacement; this function does nothing in Mac OS X.)

`SetGrowZone` (page 65) Deprecated in Mac OS X v10.4

Specifies the current heap zone's grow-zone function. (Deprecated. There is no replacement function; heaps never grow in Mac OS X, so the grow-zone function is never called.)

`TempFreeMem` (page 66) Deprecated in Mac OS X v10.4

Returns the maximum amount of free memory in the temporary heap. (Deprecated. There is no replacement function; Mac OS X does not have a separate temporary memory heap.)

`TempHLock` (page 66) Deprecated in Mac OS X v10.4

Locks a relocatable block in the temporary heap. (Deprecated. Use `HLock` (page 22) instead; Mac OS X does not have a separate temporary memory heap.)

`TempHUnlock` (page 67) Deprecated in Mac OS X v10.4

Unlocks a relocatable block in the temporary heap. (Deprecated. Use `HUnlock` (page 25) instead; Mac OS X does not have a separate temporary memory heap.)

`TempMaxMem` (page 67) Deprecated in Mac OS X v10.4
: Returns the maximum amount of temporary memory available. (Deprecated. There is no replacement function; Mac OS X does not have a separate temporary memory heap.)

`TempTopMem` (page 68) Deprecated in Mac OS X v10.4
: Returns the location of the top of the temporary heap. (Deprecated. There is no replacement function; Mac OS X does not have a separate temporary memory heap.)

`TopMem` (page 68) Deprecated in Mac OS X v10.4
: Returns a pointer to the byte at the top of an application's partition. (Deprecated. There is no replacement; this function does nothing in Mac OS X.)

`UnholdMemory` (page 69) Deprecated in Mac OS X v10.4
: Makes a currently held range of memory eligible for paging again. (Deprecated. There is no replacement; this function does nothing in Mac OS X.)

Functions

BlockMove

Copies a sequence of bytes from one location in memory to another.

```
static void BlockMove (
   const void *srcPtr,
   void *destPtr,
   Size byteCount
);
```

Parameters

srcPtr
: The address of the first byte to copy.

destPtr
: The destination address.

byteCount
: The number of bytes to copy. If the value of `byteCount` is 0, `BlockMove` does nothing.

Discussion

The `BlockMove` function copies the specified number of bytes from the address designated by srcPtr to that designated by `destPtr`. It updates no pointers.

The `BlockMove` function works correctly even if the source and destination blocks overlap.

You can safely call `BlockMove` at interrupt time. Even though it moves memory, `BlockMove` does not move relocatable blocks, but simply copies bytes.

Call the function `MemError` (page 27) to get the result code. See "Memory Manager Result Codes" (page 47).

The `BlockMove` function currently flushes the processor caches whenever it moves more than 12 bytes. This behavior can adversely affect your application's performance. You might want to avoid calling `BlockMove` to move small amounts of data in memory if there is no possibility of moving stale data or instructions. For more information about stale data and instructions, see the discussion of the processor caches in the chapter "Memory Management Utilities" in *Inside Macintosh: Memory*.

Special Considerations

Beginning in Mac OS X v10.4, the `BlockMove` function is inlined to a direct call to the POSIX `memmove` function. For more information, see the header file `MacMemory.h`.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
SoftVDigX

Declared In
`MacMemory.h`

BlockMoveData

```
static void BlockMoveData (
   const void *srcPtr,
   void *destPtr,
   Size byteCount
);
```

Parameters

srcPtr

destPtr

byteCount

Discussion
You should not make any assumptions about the state of the destination memory while `BlockMoveData` is executing. In the intermediate state, values may be present that are neither the original nor the final ones. For example, this function may use the 'dcbz' instruction. If the underlying memory is not cacheable, if the memory is write-through instead of copy-back, or if the cache block is flushed for some reason, the 'dcbz' instruction will write zeros to the destination. You can avoid the use of the 'dcbz' instruction by calling `BlockMoveDataUncached`, but even that function makes no other guarantees about the memory block's intermediate state.

Special Considerations

Beginning in Mac OS X v10.4, the `BlockMoveData` function is inlined to a direct call to the POSIX `memmove` function. For more information, see the header file `MacMemory.h`.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
QTMetaData

Declared In
`MacMemory.h`

BlockMoveDataUncached

```
static void BlockMoveDataUncached (
   const void *srcPtr,
   void *destPtr,
   Size byteCount
);
```

Parameters

srcPtr

destPtr

byteCount

Discussion

You should not make any assumptions about the state of the destination memory while `BlockMoveDataUncached` is executing. In the intermediate state, values may be present that are neither the original nor the final ones.

Special Considerations

Beginning in Mac OS X v10.4, the `BlockMoveDataUncached` function is inlined to a direct call to the POSIX `memmove` function. For more information, see the header file `MacMemory.h`.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacMemory.h`

BlockMoveUncached

```
static void BlockMoveUncached (
   const void *srcPtr,
   void *destPtr,
   Size byteCount
);
```

Parameters

srcPtr

destPtr

byteCount

Special Considerations

Beginning in Mac OS X v10.4, the `BlockMoveUncached` function is inlined to a direct call to the POSIX `memmove` function. For more information, see the header file `MacMemory.h`.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacMemory.h`

BlockZero

```
static void BlockZero (
   void *destPtr,
   Size byteCount
);
```

Parameters

destPtr

byteCount

Special Considerations

Beginning in Mac OS X v10.4, the `BlockZero` function is inlined to a direct call to the POSIX `bzero` function. For more information, see the header file `MacMemory.h`.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
Simple DrawSprocket

Declared In
`MacMemory.h`

BlockZeroUncached

```
static void BlockZeroUncached (
   void *destPtr,
   Size byteCount
);
```

Parameters

destPtr

byteCount

Special Considerations

Beginning in Mac OS X v10.4, the `BlockZeroUncached` function is inlined to a direct call to the POSIX `bzero` function. For more information, see the header file `MacMemory.h`.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

DisposeHandle

Releases memory occupied by a relocatable block.

```
void DisposeHandle (
   Handle h
);
```

Parameters

h

A handle to a relocatable block.

Discussion

The `DisposeHandle` function releases the memory occupied by the relocatable block whose handle is `h`. It also frees the handle's master pointer for other uses.

Do not use `DisposeHandle` to dispose of a handle obtained from the Resource Manager (for example, by a previous call to `GetResource`), use `ReleaseResource` instead. If, however, you have called `DetachResource` on a resource handle, you should dispose of the storage by calling `DisposeHandle`.

Call the function `MemError` (page 27) to get the result code. See "Memory Manager Result Codes" (page 47).

Special Considerations

After a call to `DisposeHandle`, all handles to the released block become invalid and should not be used again. Any subsequent calls to `DisposeHandle` using an invalid handle might crash your application.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

ASCIIMoviePlayerSample

Gamma Filter for FxPlug and AE

QTCarbonShell

QTMetaData

WhackedTV

Declared In

`MacMemory.h`

DisposePtr

Releases memory occupied by a nonrelocatable block.

```
void DisposePtr (
   Ptr p
);
```

Parameters

p

A pointer to the nonrelocatable block you want to dispose of.

Discussion

When you no longer need a nonrelocatable block, call the `DisposePtr` function to free it for other uses.

Call the function `MemError` (page 27) to get the result code. See "Memory Manager Result Codes" (page 47).

After a call to `DisposePtr`, all pointers to the released block become invalid and should not be used again. Any subsequent use of a pointer to the released block might cause a system error.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CarbonCocoa_PictureCursor
CarbonSketch
SoftVDigX

Declared In
`MacMemory.h`

EmptyHandle

Purges a relocatable block and sets the corresponding handle's master pointer to `NULL`.

```
void EmptyHandle (
   Handle h
);
```

Parameters

h

A handle to a relocatable block.

Discussion
The `EmptyHandle` function purges the relocatable block whose handle is `h` and sets the handle's master pointer to `NULL`. The `EmptyHandle` function allows you to free memory taken by a relocatable block without freeing the relocatable block's master pointer for other uses. The block whose handle is `h` must be unlocked but need not be purgeable.

Note that if there are multiple handles to the relocatable block, then calling the `EmptyHandle` function empties them all, because all of the handles share a common master pointer. When you later use `ReallocateHandle` to reallocate space for the block, the master pointer is updated, and all of the handles reference the new block correctly.

To purge all of the blocks in a heap zone that are marked purgeable, use the `PurgeMem` (page 62) function.

To free the memory taken up by a relocatable block and release the block's master pointer for other uses, use the `DisposeHandle` (page 16) function.

Call the function `MemError` (page 27) to get the result code. See "Memory Manager Result Codes" (page 47).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacMemory.h`

GetHandleSize

Returns the logical size of the relocatable block corresponding to a handle.

```
Size GetHandleSize (
   Handle h
);
```

Parameters

h

A handle to a relocatable block.

Return Value

The logical size, in bytes, of the relocatable block whose handle is h. In case of error, the function return 0.

Discussion

Call the function `MemError` (page 27) to get the result code. See "Memory Manager Result Codes" (page 47).

You should not call `GetHandleSize` at interrupt time because the heap might be in an inconsistent state.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

QTCarbonShell

SoftVDigX

Declared In

`MacMemory.h`

GetPtrSize

Returns the logical size of the nonrelocatable block corresponding to a pointer.

```
Size GetPtrSize (
   Ptr p
);
```

Parameters

p

A pointer to a nonrelocatable block.

Return Value

The logical size, in bytes, of the nonrelocatable block pointed to by p. In case of error, the function returns 0.

Discussion

Call the function `MemError` (page 27) to get the result code. See "Memory Manager Result Codes" (page 47).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MacMemory.h`

HandAndHand

Concatenates two relocatable blocks.

```
OSErr HandAndHand (
   Handle hand1,
   Handle hand2
);
```

Parameters

hand1

A handle to the first relocatable block, whose contents do not change but are concatenated to the end of the second relocatable block.

hand2

A handle to the second relocatable block, whose size the Memory Manager expands so that it can concatenate the information from `hand1` to the end of the contents of this block.

Return Value

A result code. See "Memory Manager Result Codes" (page 47).

Discussion

The `HandAndHand` function concatenates the information from the relocatable block specified by `hand1` onto the end of the relocatable block specified by `hand2`. The `hand1` variable remains unchanged.

Because the `HandAndHand` function dereferences the handle `hand1`, you must call the `HLock` function to lock the block before calling `HandAndHand`. Afterward, you can call the `HUnlock` function to unlock it. Alternatively, you can save the block's original state by calling the `HGetState` function, lock the block by calling `HLock`, and then restore the original settings by calling `HSetState`.

Because `HandAndHand` moves memory, you should not call it at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MacMemory.h`

HandToHand

Copies all of the data from one relocatable block to a new relocatable block.

```
OSErr HandToHand (
   Handle *theHndl
);
```

Parameters

theHndl

A handle to the relocatable block whose data `HandToHand` will copy. On return, `theHndl` contains a handle to a new relocatable block whose data duplicates the original.

Return Value

A result code. See "Memory Manager Result Codes" (page 47).

Discussion

The `HandToHand` function attempts to copy the information in the relocatable block to which `theHndl` is a handle; if successful, `HandToHand` sets `theHndl` to a handle pointing to the new relocatable block.

If successful in creating a new relocatable block, the HandToHand function does not duplicate the properties of the original block. The new block is unlocked, unpurgeable, and not a resource. Call HLock, HPurge, or HSetRBit (or the combination of HGetState and HSetState) to adjust the properties of the new block.

To copy only part of a relocatable block into a new relocatable block, use the PtrToHand (page 32) function. Before calling PtrToHand, lock and dereference the handle pointing to the relocatable block you want to copy.

Because HandToHand replaces its parameter with the new handle, you should retain the original parameter value somewhere else, otherwise you will not be able to access it. Here is an example:

```
Handle original, copy;
OSErr myErr;
...
copy = original;
        /*both handles access same block*/
myErr = HandToHand(copy);
        /*copy now points to copy of block*/
```

Because HandToHand allocates memory, you should not call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.

Declared In
MacMemory.h

HClrRBit

Clears the resource flag of a relocatable block.

```
void HClrRBit (
   Handle h
);
```

Parameters

h

A handle to a relocatable block. HClrRBit does nothing if the flag for the relocatable block pointed to by h is already cleared.

Discussion
The Resource Manager uses this function extensively, but you probably will not need to use it.

To disassociate the data in a resource handle from the resource file, you should use the Resource Manager function DetachResource instead of this function.

Call the function MemError (page 27) to get the result code. See "Memory Manager Result Codes" (page 47).

Availability
Available in Mac OS X v10.0 and later.

Declared In
MacMemory.h

HGetState

Returns a signed byte representing the current properties of a relocatable block.

```
SInt8 HGetState (
   Handle h
);
```

Parameters

h

A handle to a relocatable block.

Return Value

A signed byte (char) containing the flags of the master pointer for the given handle. In case of error, the value returned is meaningless.

Discussion

The `HGetState` function returns a signed byte (char) containing the flags of the master pointer for the given handle. You can save this byte, change the state of any of the flags using the functions described in this section, and then restore their original states by passing the byte to the `HSetState` function.

You can use bit-manipulation functions on the returned signed byte to determine the value of a given attribute. Currently the following bits are used:

If an error occurs during an attempt to get the state flags of the specified relocatable block, `HGetState` returns the low-order byte of the result code as its function result. For example, if the handle `h` points to a master pointer whose value is `NULL`, then the signed byte returned by `HGetState` will contain the value –109.

You may also call the function `MemError` (page 27) to get the result code. See "Memory Manager Result Codes" (page 47).

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

SoftVDigX

Declared In

`MacMemory.h`

HLock

Prevents a relocatable block from moving within its heap zone.

```
void HLock (
   Handle h
);
```

Parameters

h

A handle to a relocatable block.

Discussion

If you plan to dereference a handle and then allocate, move, or purge memory (or call a function that does so), then you should lock the handle before using the dereferenced handle.

If the block is already locked, `HLock` does nothing.

If you plan to lock a relocatable block for long periods of time, you can prevent fragmentation by ensuring that the block is as low as possible in the heap zone. To do this, see the description of the `ReserveMem` (page 64) function.

If you plan to lock a relocatable block for short periods of time, you can prevent heap fragmentation by moving the block to the top of the heap zone before locking. For more information, see the description of the `MoveHHi` (page 60) function.

Call the function `MemError` (page 27) to get the result code. See "Memory Manager Result Codes" (page 47).

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTMetaData
SoftVDigX

Declared In
`MacMemory.h`

HLockHi

Sets the lock bit on the block.

```
void HLockHi (
   Handle h
);
```

Parameters

h
A handle to a relocatable block.

Discussion
The `HLockHi` function is an alternative to using the two functions `MoveHHi` (deprecated in Mac OS X) and `HLock`. Because the `MoveHHi` function does not move memory in Mac OS X, there is no benefit to using this function.

This function will not return a meaningful error code. If you call `HLockHi` on a locked handle, it will return `noErr` (not `memLockedErr`) because it is not an error to call `HLock` on a locked handle.

Do not call `HLockHi` on blocks in the system heap. Do not call `HLockHi` from a desk accessory.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacMemory.h`

HSetRBit

Sets the resource flag of a relocatable block.

```
void HSetRBit (
   Handle h
);
```

Parameters

h

A handle to a relocatable block. HSetRBit does nothing if the flag for the relocatable block pointed to by h is already set.

Discussion
The Resource Manager uses this function extensively, but you probably will not need to use it.

When the resource flag is set, the Resource Manager identifies the associated relocatable block as belonging to a resource. This can cause problems if that block wasn't actually read from a resource.

Call the function `MemError` (page 27) to get the result code. See "Memory Manager Result Codes" (page 47).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacMemory.h`

HSetState

Restores the properties of a relocatable block.

```
void HSetState (
   Handle h,
   SInt8 flags
);
```

Parameters

h

A handle to a relocatable block.

flags

A signed byte (char) specifying the properties to which you want to set the relocatable block.

Discussion
You can use HSetState to restore properties of a block after a call to HGetState. See the description of the `HGetState` function for a list of the currently used bits in the flags byte. Because additional bits of the `flags` byte could become significant in future versions of system software, use `HSetState` only with a byte returned by `HGetState`. If you need to set two or three properties of a relocatable block at once, it is better to use the functions that set individual properties than to manipulate the bits returned by `HGetState` and then call `HSetState`.

Call the function `MemError` (page 27) to get the result code. See "Memory Manager Result Codes" (page 47).

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
SoftVDigX

Declared In
MacMemory.h

HUnlock

Allows a relocatable block to move in its heap zone.

```
void HUnlock (
   Handle h
);
```

Parameters

h

A handle to a relocatable block.

Discussion
The HUnlock function unlocks the relocatable block to which h is a handle, allowing the block to move within its heap zone. If the block is already unlocked, HUnlock does nothing.

Call the function MemError (page 27) to get the result code. See "Memory Manager Result Codes" (page 47).

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTMetaData
SoftVDigX

Declared In
MacMemory.h

IsHandleValid

Checks that a handle is valid.

```
Boolean IsHandleValid (
   Handle h
);
```

Parameters

h

The handle to check.

Return Value
Returns true if the specified handle is valid. If the handle is NULL or if the handle refers to memory which was not properly allocated, IsHandleValid returns false. In Mac OS 8 and 9, IsHandleValid also returns false if the given handle is empty. In Mac OS X, however, zero-length blocks are considered valid and IsHandleValid returns true for an empty handle.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacMemory.h`

IsHeapValid

Always returns true in Mac OS X.

```
Boolean IsHeapValid (
   void
);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacMemory.h`

IsPointerValid

Checks that a pointer is valid.

```
Boolean IsPointerValid (
   Ptr p
);
```

Parameters

p

The pointer to check.

Return Value
Returns `true` if the specified pointer is valid. If the pointer is `NULL` or if the pointer points to memory which was not properly allocated, `IsPointerValid` returns `false`. In Mac OS 8 and 9, `IsPointerValid` also returns `false` if the given pointer points to a zero-length block in memory. In Mac OS X, however, zero-length blocks are considered valid and `IsPointerValid` returns `true`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacMemory.h`

LMGetMemErr

Returns the result of the last Memory Manager function without clearing the value.

```
SInt16 LMGetMemErr (
   void
);
```

Return Value
A result code. See "Memory Manager Result Codes" (page 47).

Availability
Available in Mac OS X v10.0 and later.

Declared In
MacMemory.h

LMSetMemErr

Sets the value which will be returned by the MemError function.

```
void LMSetMemErr (
   SInt16 value
);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
MacMemory.h

MemError

Determines if an application's last direct call to a Memory Manager function executed successfully.

```
OSErr MemError (
   void
);
```

Return Value
A result code. See "Memory Manager Result Codes" (page 47).

Discussion
For each thread, MemError yields the result code produced by the last Memory Manager function your application called directly.

MemError is useful during application debugging. You might also use MemError as one part of a memory-management scheme to identify instances in which the Memory Manager rejects overly large memory requests by returning the error code memFullErr.

To view the result codes that MemError can produce, see "Memory Manager Result Codes" (page 47).

Do not rely on MemError as the only component of a memory-management scheme. For example, suppose you call NewHandle or NewPtr and receive the result code noErr, indicating that the Memory Manager was able to allocate sufficient memory. In this case, you have no guarantee that the allocation did not deplete your application's memory reserves to levels so low that simple operations might cause your application to crash. Instead of relying on MemError, check before making a memory request that there is enough memory both to fulfill the request and to support essential operations.

Version Notes
Starting with Mac OS X v10.3, the MemError function provides error codes on a per-thread basis.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTCarbonShell
SoftVDigX

Declared In
`MacMemory.h`

NewEmptyHandle

Initializes a new handle without allocating any memory for it to control.

```
Handle NewEmptyHandle (
   void
);
```

Return Value
A handle with its master pointer set to `NULL`.

Discussion
The Resource Manager uses this function extensively, but you probably will not need to use it.

When you want to allocate memory for the empty handle, use the `ReallocateHandle` (page 33) function.

Call the function `MemError` (page 27) to get the result code. See "Memory Manager Result Codes" (page 47).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacMemory.h`

NewHandle

Allocates a new relocatable memory block of a specified size in the current heap zone.

```
Handle NewHandle (
   Size byteCount
);
```

Parameters

`byteCount`

The requested size, in bytes, of the relocatable block. Maximum size is 2 GB, the maximum size for variables of type `Size`.

Return Value
A handle to the new block. If `NewHandle` cannot allocate a block of the requested size, it returns `NULL`.

Discussion
The `NewHandle` function pursues all available avenues to create a block of the requested size, including compacting the heap zone, increasing its size, and purging blocks from it. If all of these techniques fail and the heap zone has a grow-zone function installed, `NewHandle calls the function`. Then `NewHandle`

tries again to free the necessary amount of memory, once more compacting and purging the heap zone if necessary. If `NewHandle` still cannot allocate memory, `NewHandle` calls the grow-zone function again, unless that function had returned 0, in which case `NewHandle` gives up and returns `NULL`.

If the `NewHandle` function succeeds in creating the requested block, this new block is unlocked and unpurgeable.

If you allocate a relocatable block that you plan to lock for long periods of time, you can prevent heap fragmentation by allocating the block as low as possible in the heap zone. To do this, see the description of the function `ReserveMem` (page 64).

If you plan to lock a relocatable block for short periods of time, you might want to move it to the top of the heap zone to prevent heap fragmentation. For more information, see the description of the function `MoveHHi` (page 60).

Call the function `MemError` (page 27) to get the result code. See "Memory Manager Result Codes" (page 47).

Because `NewHandle` allocates memory, you should not call it at interrupt time.

Do not try to manufacture your own handles without this function by simply assigning the address of a variable of type `Ptr` to a variable of type `Handle`. The resulting "fake handle" would not reference a relocatable block and could cause a system crash.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Gamma Filter for FxPlug and AE
QTCarbonShell
QTMetaData
WhackedTV

Declared In
`MacMemory.h`

NewHandleClear

Allocates a relocatable block of memory of a specified size with all its bytes set to 0.

```
Handle NewHandleClear (
   Size byteCount
);
```

Parameters

byteCount
The requested size (in bytes) of the relocatable block. The `NewHandleClear` function sets each of these bytes to 0.

Return Value
A handle to the new block. If `NewHandleClear` cannot allocate a block of the requested size, it returns `NULL`.

Discussion
The `NewHandleClear` function works like the `NewHandle` function, but sets all bytes in the new block to 0 instead of leaving the contents of the block undefined.

Call the function `MemError` (page 27) to get the result code. See "Memory Manager Result Codes" (page 47).

Because `NewHandleClear` allocates memory, you should not call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
LiveVideoMixer2
SoftVDigX

Declared In
`MacMemory.h`

NewPtr

Allocates a nonrelocatable block of memory of a specified size.

```
Ptr NewPtr (
   Size byteCount
);
```

Parameters

byteCount

The requested size (in bytes) of the nonrelocatable block. In Mac OS X, if you pass a value of zero, this function returns NULL, and `MemError` is set to `paramErr`. In Mac OS 9 and earlier, if you pass a value of zero, this function returns a valid zero length pointer.

Return Value
A pointer to the new block. If `NewPtr` fails to allocate a block of the requested size, it returns `NULL`.

Discussion
The `NewPtr` function attempts to reserve space as low in the heap zone as possible for the new block. If it is able to reserve the requested amount of space, `NewPtr` allocates the nonrelocatable block in the gap `ReserveMem` creates. Otherwise, `NewPtr` returns `NULL` and generates a `memFullErr` error.

Call the function `MemError` (page 27) to get the result code. See "Memory Manager Result Codes" (page 47).

Because `NewPtr` allocates memory, you should not call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacMemory.h`

NewPtrClear

Allocates a nonrelocatable block of memory of a specified size with all its bytes set to 0.

```
Ptr NewPtrClear (
   Size byteCount
);
```

Parameters

byteCount

The requested size (in bytes) of the nonrelocatable block.

Return Value

A pointer to the new block. If NewPtrClear fails to allocate a block of the requested size, it returns `NULL`.

Discussion

The `NewPtrClear` function works much as the `NewPtr` function does, but sets all bytes in the new block to 0 instead of leaving the contents of the block undefined.

Call the function `MemError` (page 27) to get the result code. See "Memory Manager Result Codes" (page 47).

Because `NewPtrClear` allocates memory, you should not call it at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CarbonCocoa_PictureCursor

CarbonSketch

SoftVDigX

Declared In

`MacMemory.h`

PtrAndHand

Concatenates part or all of a memory block to the end of a relocatable block.

```
OSErr PtrAndHand (
   const void *ptr1,
   Handle hand2,
   long size
);
```

Parameters

ptr1

A pointer to the beginning of the data that the Memory Manager is to concatenate onto the end of the relocatable block.

hand2

A handle to the relocatable block, whose size the Memory Manager expands so that it can concatenate the information from ptr1 onto the end of this block.

size

The number of bytes of the block referenced by ptr1 to copy.

Return Value
A result code. See "Memory Manager Result Codes" (page 47).

Discussion
The `PtrAndHand` function takes the number of bytes specified by the `size` parameter, beginning at the location specified by `ptr1`, and concatenates them onto the end of the relocatable block to which `hand2` is a handle. The contents of the source block remain unchanged.

Because `PtrAndHand` allocates memory, you should not call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTCarbonShell

Declared In
`MacMemory.h`

PtrToHand

Copies data referenced by a pointer to a new relocatable block.

```
OSErr PtrToHand (
   const void *srcPtr,
   Handle *dstHndl,
   long size
);
```

Parameters

srcPtr
The address of the first byte to copy.

dstHndl
A handle for which you have not yet allocated any memory. The `PtrToHand` function allocates memory for the handle and copies the specified number of bytes beginning at `srcPtr` into it. The dstHndl parameter is an output parameter that will hold the result. Its value on entry is ignored. If no error occurs, on exit it points to an unlocked, non-purgeable Handle of the requested size.

size
The number of bytes to copy.

Return Value
A result code. See "Memory Manager Result Codes" (page 47).

Discussion
If you dereference and lock a handle, the `PtrToHand` function can copy its data to a new handle. However, for copying data from one handle to another, the `HandToHand` (page 20) function is more efficient.

Because `PtrToHand` allocates memory, you should not call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacMemory.h`

PtrToXHand

Copies data referenced by a pointer to an existing relocatable block.

```
OSErr PtrToXHand (
   const void *srcPtr,
   Handle dstHndl,
   long size
);
```

Parameters

srcPtr

The address of the first byte to copy.

dstHndl

A handle to an existing relocatable block.

size

The number of bytes to copy.

Return Value

A result code. See "Memory Manager Result Codes" (page 47).

Discussion

The `PtrToXHand` function copies the specified number of bytes from the location specified by `srcPtr` to the handle specified by `dstHndl`.

Because `PtrToXHand` affects memory, you should not call it at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MacMemory.h`

ReallocateHandle

Allocates a new relocatable block of a specified size and sets a handle's master pointer to point to the new block.

```
void ReallocateHandle (
   Handle h,
   Size byteCount
);
```

Parameters

h

A handle to a relocatable block.

byteCount

The desired new logical size (in bytes) of the relocatable block. The new block is unlocked and unpurgeable.

Discussion

Usually you use `ReallocateHandle` to reallocate space for a block that you have emptied or the Memory Manager has purged. If the handle references an existing block, `ReallocateHandle` releases that block before creating a new one.

If many handles reference a single purged, relocatable block, you need to call `ReallocateHandle` on just one of them.

To reallocate space for a resource that has been purged, you should call `LoadResource`, not `ReallocateHandle`. To resize relocatable blocks, you should call the `SetHandleSize` (page 35) function.

Currently in Mac OS 8 and 9, the `ReallocateHandle` function releases any existing relocatable block referenced by the handle `h` before allocating a new one. This behavior means that if an error occurs when calling `ReallocateHandle`, the handle `h` will be set to `NULL`. This behavior does not occur in the Mac OS X implementation.

Call the function `MemError` (page 27) to get the result code. See "Memory Manager Result Codes" (page 47).

Because `ReallocateHandle` might purge and allocate memory, you should not call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacMemory.h`

RecoverHandle

Returns a handle to a relocatable block pointed to by a specified `pointer`.

```
Handle RecoverHandle (
   Ptr p
);
```

Parameters

p

The master pointer to a relocatable block.

Return Value
A handle to a relocatable block point to by p. If `p` does not point to a valid block, the results of `RecoverHandle` are undefined.

Discussion
The Memory Manager does not allow you to change relocatable blocks into nonrelocatable blocks, or vice-versa. However, if you no longer have access to a handle but still have access to its master pointer p, you can use the `RecoverHandle` function to recreate a handle to the relocatable block referenced by p.

Call the function `MemError` (page 27) to get the result code. See "Memory Manager Result Codes" (page 47).

Even though `RecoverHandle` does not move or purge memory, you should not call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacMemory.h`

SetHandleSize

Changes the logical size of the relocatable block corresponding to the specified handle.

```
void SetHandleSize (
   Handle h,
   Size newSize
);
```

Parameters

h

A handle to a relocatable block.

newSize

The desired new logical size, in bytes, of the relocatable block.

Discussion

`SetHandleSize` tries to change the size of the allocation to *newSize*. If there is not enough room to enlarge the memory allocation pointed to by *h*, `SetHandleSize` creates a new allocation, copies as much of the old data pointed to by *h* as will fit to the new allocation, and frees the old allocation. `SetHandleSize` might need to move the relocatable block to obtain enough space for the resized block. Thus, for best results you should unlock a block before resizing it.

An attempt to increase the size of a locked block might fail, because of blocks above and below it that are either nonrelocatable or locked. You should be prepared for this possibility.

Instead of using the `SetHandleSize` function to set the size of a handle to `0`, you can use the `EmptyHandle` (page 18) function.

Call the function `MemError` (page 27) to get the result code. See "Memory Manager Result Codes" (page 47).

Because `SetHandleSize` allocates memory, you should not call it at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

SoftVDigX

Declared In

`MacMemory.h`

SetPtrSize

Changes the logical size of the nonrelocatable block corresponding to a pointer.

```
void SetPtrSize (
   Ptr p,
   Size newSize
);
```

Parameters

p

A pointer to a nonrelocatable block.

newSize

The desired new logical size, in bytes, of the nonrelocatable block.

Discussion
An attempt to increase the size of a nonrelocatable block might fail because of a block above it that is either nonrelocatable or locked. You should be prepared for this possibility.

Call the function `MemError` (page 27) to get the result code. See "Memory Manager Result Codes" (page 47).

Because `SetPtrSize` allocates memory, you should not call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacMemory.h`

TempNewHandle

Allocates a new relocatable block of temporary memory.

```
Handle TempNewHandle (
   Size logicalSize,
   OSErr *resultCode
);
```

Parameters

logicalSize

The requested logical size, in bytes, of the new temporary block of memory.

resultCode

On return, the result code from the function call. See "Memory Manager Result Codes" (page 47).

Return Value
A handle to a block of size `logicalSize`. If it cannot allocate a block of that size, the function returns `NULL`.

Discussion
Before calling `TempNewHandle`, you should call `TempFreeMem` or `TempMaxMem` to make sure that there is enough free space to satisfy the request.

Because `TempNewHandle` might allocate memory, you should not call it at interrupt time.

Carbon Porting Notes

Temporary memory allocations will actually come from the applications's address space in Mac OS X. However, Carbon applications running under Mac OS 8.x will be able to get true temporary memory.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacMemory.h`

Callbacks

All Memory Manager callbacks are deprecated in Mac OS X. They are non-functional.

GrowZoneProcPtr

Deprecated.

```
typedef long (*GrowZoneProcPtr) (
    Size cbNeeded
);
```

If you name your function `MyGrowZoneProc`, you would declare it like this:

```
long MyGrowZoneProc (
    Size cbNeeded
);
```

Parameters

cbNeeded

The physical size, in bytes, of the needed block, including the block header. The grow-zone function should attempt to create a free block of at least this size.

Return Value

The number of bytes of memory the function has freed.

Discussion

User-defined function that creates free space in the heap.

Whenever the Memory Manager has exhausted all available means of creating space within your application heap—including purging, compacting, and (if possible) expanding the heap—it calls your application-defined grow-zone function. The grow-zone function can do whatever is necessary to create free space in the heap. Typically, a grow-zone function marks some unneeded blocks as purgeable or releases an emergency memory reserve maintained by your application.

The grow-zone function should return a nonzero value equal to the number of bytes of memory it has freed, or zero if it is unable to free any. When the function returns a nonzero value, the Memory Manager once again purges and compacts the heap zone and tries to reallocate memory. If there is still insufficient memory, the Memory Manager calls the grow-zone function again (but only if the function returned a nonzero value the previous time it was called). This mechanism allows your grow-zone function to release just a little bit of memory at a time. If the amount it releases at any time is not enough, the Memory Manager calls it again and gives it the opportunity to take more drastic measures.

The Memory Manager might designate a particular relocatable block in the heap as protected; your grow-zone function should not move or purge that block. You can determine which block, if any, the Memory Manager has protected by calling the `GZSaveHnd` function in your grow-zone function.

Remember that the Memory Manager calls a grow-zone function while attempting to allocate memory. As a result, your grow-zone function should not allocate memory itself or perform any other actions that might indirectly cause memory allocation (such as calling functions in unloaded code segments or displaying dialog boxes).

You install a grow-zone function by passing its address to the `InitZone` function when you create a new heap zone or by calling the `SetGrowZone` function at any other time.

Your grow-zone function might be called at a time when the system is attempting to allocate memory and the value in the A5 register is not correct. If your function accesses your application's A5 world or makes any trap calls, you need to set up and later restore the A5 register by calling `SetCurrentA5` and `SetA5`. See the chapter "Memory Management Utilities" in this book for a description of these two functions.

Because of the optimizations performed by some compilers, the actual work of the grow-zone function and the setting and restoring of the A5 register might have to be placed in separate functions.

See the chapter "Introduction to Memory Management" for a definition of a sample grow-zone function.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

PurgeProcPtr

Deprecated.

```
typedef void (*PurgeProcPtr) (
    Handle blockToPurge
);
```

If you name your function `MyPurgeProc`, you would declare it like this:

```
void MyPurgeProc (
    Handle blockToPurge
);
```

Parameters

`blockToPurge`
A handle to the block that is about to be purged.

Discussion
User-defined function called when the Memory Manager needs to purge a block or allocate memory.

Whenever the Memory Manager needs to purge a block from the application heap, it first calls any application-defined purge-warning function that you have installed. The purge-warning function can, if necessary, save the contents of that block or otherwise respond to the warning.

Your purge-warning function is called during a memory-allocation request. As a result, you should not call any functions that might cause memory to be moved or purged. In particular, if you save the data of the block in a file, the file should already be open when your purge-warning function is called, and you should write the data synchronously.

You should not dispose of or change the purgeable status of the block whose handle is passed to your function.

To install a purge-warning function, you need to assign its address to the `purgeProc` field of the associated zone header.

Note that if you call the Resource Manager function `SetResPurge` with the parameter `TRUE`, any existing purge-warning function is replaced by a purge-warning function installed by the Resource Manager. You can execute both warning functions by calling `SetResPurge`, saving the existing value of the `purgeProc` field of the zone header, and then reinstalling your purge-warning function. Your purge-warning function should call the Resource Manager's purge-warning function internally.

Your purge-warning function might be called at a time when the system is attempting to allocate memory and the value in the A5 register is not correct. If your function accesses your application's A5 world or makes any trap calls, you need to set up and later restore the A5 register by calling `SetCurrentA5` and `SetA5`.

Because of the optimizations performed by some compilers, the actual work of the purge-warning function and the setting and restoring of the A5 register might have to be placed in separate functions.

The Memory Manager calls your purge-warning function for every handle that is about to be purged (not necessarily for every purgeable handle in your heap, however). Your function should be able to determine quickly whether the handle that the Memory Manager is about to purge points to data you need to save or otherwise process.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

UserFnProcPtr

Deprecated.

```
typedef void (*UserFnProcPtr) (
    void *parameter
);
```

If you name your function `MyUserFnProc`, you would declare it like this:

```
void MyUserFnProc (
    void * parameter
);
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

Data Types

All Memory Manager data types are deprecated in Mac OS X. They are not used.

BackingFileID

Deprecated.

```
typedef struct * BackingFileID;
```

Special Considerations

FileViewAccess

Deprecated.

```
typedef UInt32 FileViewAccess;
enum {
    kFileViewAccessReadBit = 0,
    kFileViewAccessWriteBit = 1,
    kFileViewAccessExecuteBit = 2,
    kFileViewAccessReadMask = 1,
    kFileViewAccessWriteMask = 2,
    kFileViewAccessExecuteMask = 4,
    kFileViewAccessExcluded = 0,
    kFileViewAccessReadOnly = 5,
    kFileViewAccessReadWrite = 7
};
```

Special Considerations

FileViewID

Deprecated.

```
typedef struct * FileViewID;
```

Special Considerations

FileViewInformation

Deprecated.

```
struct FileViewInformation {
    ProcessSerialNumber owningProcess;
    LogicalAddress viewBase;
    ByteCount viewLength;
    BackingFileID backingFile;
    UInt64 backingBase;
    FileViewAccess access;
    ByteCount guardLength;
    FileViewOptions options;
};
```

FileViewOptions

Deprecated.

```
typedef OptionBits FileViewOptions;
```

GrowZoneUPP

Deprecated.

```
typedef GrowZoneProcPtr GrowZoneUPP;
```

Discussion
For more information, see the description of the GrowZoneUPP () callback function.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

LogicalToPhysicalTable

Deprecated.

```
struct LogicalToPhysicalTable {
    MemoryBlock logical;
    MemoryBlock physical[8];
};
typedef struct LogicalToPhysicalTable LogicalToPhysicalTable;
```

Fields

`logical`
: A logical block of memory whose corresponding physical blocks are to be determined.

`physical`
: A physical translation table that identifies the blocks of physical memory corresponding to the logical block identified in the `logical` field.

Discussion
The `GetPhysical` function uses a translation table to hold information about a logical address range and its corresponding physical addresses. A translation table is defined by the data type LogicalToPhysicalTable.

Special Considerations

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

MappedFileAttributes

Deprecated.

```
typedef UInt32 MappedFileAttributes;
enum {
    kIsMappedScratchFile = 1
};
```

MappedFileInformation

Deprecated.

```
struct MappedFileInformation {
    ProcessSerialNumber owningProcess;
    FSRef *ref;
    HFSUniStr255 *forkName;
    MappingPrivileges privileges;
    UInt64 currentSize;
    MappedFileAttributes attributes;
};
```

MappingPrivileges

Deprecated.

```
typedef UInt32 MappingPrivileges;
enum {
    kInvalidMappedPrivileges = 0,
    kCanReadMappedFile = 1,
    kCanWriteMappedFile = 2,
    kNoProcessMappedFile = -2147483648,
    kValidMappingPrivilegesMask = -2147483645
};
```

Special Considerations

MemoryBlock

Deprecated.

```
struct MemoryBlock {
    void * address;
    unsigned long count;
};
typedef struct MemoryBlock MemoryBlock;
```

Fields

`address`

A pointer to the beginning of a block of memory.

`count`

The number of bytes in the block of memory.

Discussion

The `GetPhysical` function uses a structure of type MemoryBlock to hold information about a block of memory, either logical or physical.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacMemory.h`

PurgeUPP

Deprecated.

```
typedef PurgeProcPtr PurgeUPP;
```

Discussion

For more information, see the description of the PurgeUPP () callback function.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacMemory.h`

StatusRegisterContents

Deprecated.

```
typedef  StatusRegisterContents;
```

Special Considerations

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacMemory.h`

UserFnUPP

Deprecated.

```
typedef UserFnProcPtr UserFnUPP;
```

Discussion
For more information, see the description of the UserFnUPP () callback function.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
MacMemory.h

VolumeVirtualMemoryInfo

Deprecated.

```
struct VolumeVirtualMemoryInfo {
    PBVersion version;
    SInt16 volumeRefNum;
    Boolean inUse;
    UInt8 _fill;
    UInt32 vmOptions;
};
typedef struct VolumeVirtualMemoryInfo VolumeVirtualMemoryInfo;
typedef VolumeVirtualMemoryInfo * VolumeVirtualMemoryInfoPtr;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
MacMemory.h

Zone

Deprecated.

```
struct Zone {
    Ptr bkLim;
    Ptr purgePtr;
    Ptr hFstFree;
    long zcbFree;
    GrowZoneUPP gzProc;
    short moreMast;
    short flags;
    short cntRel;
    short maxRel;
    short cntNRel;
    SInt8 heapType;
    SInt8 unused;
    short cntEmpty;
    short cntHandles;
    long minCBFree;
    PurgeUPP purgeProc;
    Ptr sparePtr;
    Ptr allocPtr;
    short heapData;
};
typedef struct Zone Zone;
typedef Zone * THz;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

Constants

All Memory Manager constants are deprecated in Mac OS X. They are not used.

Default Physical Entry Count Constant

Deprecated.

```
enum {
    defaultPhysicalEntryCount = 8
};
```

Discussion
The defaultPhysicalEntryCount constant represents the default number of physical blocks in a table.

k32BitHeap

Deprecated.

```
enum {
    k32BitHeap = 1,
    kNewStyleHeap = 2,
    kNewDebugHeap = 4
};
```

kFileViewInformationVersion1

Deprecated.

```
enum {
    kFileViewInformationVersion1 = 1
};
```

kHandleIsResourceBit

Deprecated.

```
enum {
    kHandleIsResourceBit = 5,
    kHandlePurgeableBit = 6,
    kHandleLockedBit = 7
};
```

kHandleIsResourceMask

Deprecated.

```
enum {
    kHandleIsResourceMask = 0x20,
    kHandlePurgeableMask = 0x40,
    kHandleLockedMask = 0x80
};
```

kMapEntireFork

Deprecated.

```
enum {
    kMapEntireFork = -1
};
```

Constants

`kMapEntireFork`

kMappedFileInformationVersion1

Deprecated.

```
enum {
    kMappedFileInformationVersion1 = 1
};
```

kPageInMemory

Deprecated.

```
typedef short PageState;
enum {
    kPageInMemory = 0,
    kPageOnDisk = 1,
    kNotPaged = 2
};
```

Discussion
The `GetPageState` function obtains the state value of a page of logical memory. The `PageState` data type defines constants that represent these possible state values.

Debuggers need a way to display the contents of memory without paging or to display the contents of pages currently on disk. The `GetPageState` function obtains a constant from the PageState data type to specify the state of a page containing a virtual address. A debugger can use this information to determine whether certain memory addresses should be referenced. Note that ROM and I/O space are not pageable and therefore are considered not paged.

kVolumeVirtualMemoryInfoVersion1

Deprecated.

```
enum {
    kVolumeVirtualMemoryInfoVersion1 = 1
};
```

maxSize

Deprecated.

```
enum {
    maxSize = 0x7FFFFFF0
};
```

Result Codes

The most common result codes returned by the Memory Manager are listed below.

Result Code	Value	Description
`menuPrgErr`	84	A menu was purged. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
`negZcbFreeErr`	33	A heap has been corrupted. Available in Mac OS X v10.0 and later.
`memROZErr`	-99	Operation on a read-only zone. This result code is not relevant in Mac OS X. Available in Mac OS X v10.0 and later.
`memFullErr`	-108	Not enough memory in heap. Available in Mac OS X v10.0 and later.
`nilHandleErr`	-109	Handle argument is `NULL`. Available in Mac OS X v10.0 and later.
`memAdrErr`	-110	Address is odd or out of range. Available in Mac OS X v10.0 and later.
`memWZErr`	-111	Attempt to operate on a free block. Available in Mac OS X v10.0 and later.
`memPurErr`	-112	Attempt to purge a locked or unpurgeable block. Available in Mac OS X v10.0 and later.
`memAZErr`	-113	Address in zone check failed. Available in Mac OS X v10.0 and later.
`memPCErr`	-114	Pointer check failed. Available in Mac OS X v10.0 and later.
`memBCErr`	-115	Block check failed. Available in Mac OS X v10.0 and later.
`memSCErr`	-116	Size check failed. Available in Mac OS X v10.0 and later.
`memLockedErr`	-117	Block is locked. Available in Mac OS X v10.0 and later.

Deprecated Memory Manager Functions

A function identified as deprecated has been superseded and may become unsupported in the future.

Deprecated in Mac OS X v10.4

CheckAllHeaps

Checks all known heaps for validity. (Deprecated in Mac OS X v10.4. There is no replacement function; an application has access only to its own heap in Mac OS X.)

```
Boolean CheckAllHeaps (
   void
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

CompactMem

Compacts the heap by moving relocatable blocks as needed. (Deprecated in Mac OS X v10.4. There is no replacement function; memory compaction is never needed and never performed in Mac OS X.)

```
Size CompactMem (
   Size cbNeeded
);
```

Parameters
cbNeeded
The size, in bytes, of the block for which `CompactMem` should attempt to make room.

Return Value
The size, in bytes, of the largest contiguous free block available after compacting the heap zone. CompactMem does not actually allocate that block.

Discussion
The Memory Manager automatically compacts the heap when a memory request fails. However, you can use the `CompactMem` function to compact the current heap zone manually.

`CompactMem` compacts the current heap zone not by purging blocks, but rather by moving unlocked, relocatable blocks down until they encounter nonrelocatable blocks or locked, relocatable blocks. `CompactMem` continues compacting until it either finds a contiguous block of at least `cbNeeded` free bytes or compacts the entire zone.

To compact the entire heap zone, call `CompactMem(maxSize)`.

Call the function `MemError` (page 27) to get the result code. See "Memory Manager Result Codes" (page 47).

Because `CompactMem` moves memory, you should not call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

DisposeGrowZoneUPP

(Deprecated in Mac OS X v10.4. There is no replacement function; heaps never grow in Mac OS X, so the grow-zone function is never called.)

```
void DisposeGrowZoneUPP (
   GrowZoneUPP userUPP
);
```

Parameters
userUPP

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

DisposePurgeUPP

(Deprecated in Mac OS X v10.4. There is no replacement function; heaps are never purged in Mac OS X, so the purge function is never called.)

```
void DisposePurgeUPP (
   PurgeUPP userUPP
);
```

Parameters
userUPP

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

DisposeUserFnUPP

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
void DisposeUserFnUPP (
   UserFnUPP userUPP
);
```

Parameters
userUPP

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

FlushMemory

Makes a portion of the address space clean. (Deprecated in Mac OS X v10.4. There is no replacement; this function does nothing in Mac OS X.)

```
OSErr FlushMemory (
   void *address,
   unsigned long count
);
```

Return Value
This function always returns a value of `noErr`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

GetGrowZone

Returns the current heap zone's grow-zone function. (Deprecated in Mac OS X v10.4. There is no replacement function; heaps never grow in Mac OS X, so the grow-zone function is never used.)

```
GrowZoneUPP GetGrowZone (
   void
);
```

Return Value
See the description of the `GrowZoneUPP` data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

GZSaveHnd

Returns a relocatable block to be protected during grow-zone operations. (Deprecated in Mac OS X v10.4. There is no replacement; this function does nothing in Mac OS X.)

```
Handle GZSaveHnd (
   void
);
```

Return Value
A handle to a block of memory that the Memory Manager reserves during grow-zone operations. Your grow-zone function must not move, purge, or delete this block. This function returns `NULL` if there is no such block.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

HNoPurge

Marks a relocatable block as unpurgeable. (Deprecated in Mac OS X v10.4. There is no replacement function; heaps are never purged in Mac OS X.)

```
void HNoPurge (
   Handle h
);
```

Parameters

h

A handle to a relocatable block.

Discussion
The `HNoPurge` function marks the relocatable block, to which `h` is a handle, as unpurgeable. If the block is already unpurgeable, `HNoPurge` does nothing.

The `HNoPurge` function does not reallocate memory for a handle if it has already been purged.

If you want to reallocate memory for a relocatable block that has already been purged, you can use the `ReallocateHandle` (page 33) function.

Call the function `MemError` (page 27) to get the result code. See "Memory Manager Result Codes" (page 47).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

HoldMemory

Makes a portion of the address space resident in physical memory and ineligible for paging. (Deprecated in Mac OS X v10.4. There is no replacement; this function does nothing in Mac OS X.)

```
OSErr HoldMemory (
   void *address,
   unsigned long count
);
```

Parameters

address
A pointer indicating the starting address of the range of memory to be held in RAM.

count
The size, in bytes, of the range of memory to be held in RAM.

Return Value
This function always returns a value of `noErr`.

Discussion
If the starting address you supply to the `HoldMemory` function is not on a page boundary, then `HoldMemory` rounds down to the nearest page boundary. Similarly, if the specified range does not end on a page boundary, `HoldMemory` rounds up the value you pass in the `count` parameter so that the entire range of memory is held.

Even though `HoldMemory` does not move or purge memory, you should not call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

HPurge

Marks a relocatable block as purgeable. (Deprecated in Mac OS X v10.4. There is no replacement function; heaps are never purged in Mac OS X.)

```
void HPurge (
   Handle h
);
```

Parameters

h

A handle to a relocatable block.

Discussion

The `HPurge` function marks the relocatable block, to which `h` is a handle, as purgeable. If the block is already purgeable, `HPurge` does nothing.

The Memory Manager might purge the block when it needs to purge the heap zone containing the block to satisfy a memory request. A direct call to the `MaxMem` function would also purge blocks marked as purgeable.

Once you mark a relocatable block as purgeable, you should make sure that handles to the block are not empty before you access the block. If they are empty, you must reallocate space for the block and recopy the block's data from another source, such as a resource file, before using the information in the block.

If the block to which `h` is a handle is locked, `HPurge` does not unlock the block but does mark it as purgeable. If you later call `HUnlock` on `h`, the block is subject to purging.

If the Memory Manager has purged a block, you can reallocate space for it by using the `ReallocateHandle` (page 33) function.

You can immediately free the space taken by a handle without disposing of it by calling the function `EmptyHandle` (page 18). This function does not require that the block be purgeable.

Call the function `MemError` (page 27) to get the result code. See "Memory Manager Result Codes" (page 47).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`MacMemory.h`

InvokeGrowZoneUPP

(Deprecated in Mac OS X v10.4. There is no replacement function; heaps never grow in Mac OS X, so the grow-zone function is never called.)

```
long InvokeGrowZoneUPP (
   Size cbNeeded,
   GrowZoneUPP userUPP
);
```

Parameters

cbNeeded

userUPP

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

MacMemory.h

InvokePurgeUPP

(Deprecated in Mac OS X v10.4. There is no replacement function; heaps are never purged in Mac OS X, so the purge function is never called.)

```
void InvokePurgeUPP (
   Handle blockToPurge,
   PurgeUPP userUPP
);
```

Parameters

blockToPurge

userUPP

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

MacMemory.h

InvokeUserFnUPP

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
void InvokeUserFnUPP (
   void *parameter,
   UserFnUPP userUPP
);
```

Parameters

parameter

userUPP

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
MacMemory.h

LMGetApplZone

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
THz LMGetApplZone (
   void
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
MacMemory.h

LMGetSysZone

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
THz LMGetSysZone (
   void
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
MacMemory.h

LMSetApplZone

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
void LMSetApplZone (
   THz value
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

LMSetSysZone

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
void LMSetSysZone (
   THz value
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

MakeMemoryNonResident

Makes pages in the specified range immediately available for reuse. (Deprecated in Mac OS X v10.4. There is no replacement; this function does nothing in Mac OS X.)

```
OSErr MakeMemoryNonResident (
   void *address,
   unsigned long count
);
```

Return Value
This function always returns a value of `noErr`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
MacMemory.h

MakeMemoryResident

Makes a portion of the address space resident in physical memory. (Deprecated in Mac OS X v10.4. There is no replacement; this function does nothing in Mac OS X.)

```
OSErr MakeMemoryResident (
   void *address,
   unsigned long count
);
```

Return Value
A result code. See "Memory Manager Result Codes" (page 47).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
MacMemory.h

MoreMasterPointers

Allocates a specified number of master pointers in the current heap zone. (Deprecated in Mac OS X v10.4. There is no replacement function; master pointers do not need to be pre-allocated in Mac OS X.)

```
void MoreMasterPointers (
   UInt32 inCount
);
```

Parameters
inCount
The number of master pointers you want to allocate in a single nonrelocatable block.

Carbon Porting Notes

Carbon applications should use this function instead of `MoreMasters` to allocate a nonrelocatable block of master pointers in the current heap zone.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
MacMemory.h

MoreMasters

Allocates a block of master pointers in the current heap zone. (Deprecated in Mac OS X v10.4. There is no replacement function; master pointers do not need to be pre-allocated in Mac OS X.)

```
void MoreMasters (
   void
);
```

Discussion
In the application heap, a block of master pointers consists of 64 master pointers, and in the system heap, a block consists of 32 master pointers. (These values are likely to increase in future versions of system software.) When you initialize additional heap zones, you can specify the number of master pointers you want to have in a block of master pointers.

The Memory Manager automatically calls the `MoreMasters` function once for every new heap zone, including the application heap zone.

Call `MoreMasters` several times at the beginning of your program to prevent the Memory Manager from running out of master pointers in the middle of application execution. If it does run out, it allocates more, possibly causing heap fragmentation.

You should call `MoreMasters` at the beginning of your program enough times to ensure that the Memory Manager never needs to call it for you. For example, if your application never allocates more than 300 relocatable blocks in its heap zone, then five calls to the `MoreMasters` should be enough. It's better to call `MoreMasters` too many times than too few. For instance, if your application usually allocates about 100 relocatable blocks but might allocate 1000 in a particularly busy session, call `MoreMasters` enough times to accommodate the largest amount.

If you initialize a new zone, you can specify the number of master pointers that a master pointer block should contain.

Call the `MemError` (page 27) function to get the result code. See "Memory Manager Result Codes" (page 47).

Because `MoreMasters` allocates memory, you should not call it at interrupt time.

The calls to `MoreMasters` at the beginning of your application should be in the main code segment of your application or in a segment that the main segment never unloads.

Carbon Porting Notes

You should instead use `MoreMasterPointers` (page 58).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Related Sample Code
Simple DrawSprocket

Declared In
`MacMemory.h`

MoveHHi

Moves a relocatable block as high in memory as possible. (Deprecated in Mac OS X v10.4. There is no replacement function; there is no benefit to moving handles high in memory in Mac OS X.)

```
void MoveHHi (
   Handle h
);
```

Parameters

h

A handle to a relocatable block.

Discussion

This function moves a relocatable block as high in memory as possible to help prevent heap fragmentation. The `MoveHHi` function attempts to move the relocatable block referenced by the handle `h` upward until it reaches a nonrelocatable block, a locked relocatable block, or the top of the heap.

If you plan to lock a relocatable block for a short period of time, use the `MoveHHi` function, which moves the block to the top of the heap and thus helps prevent heap fragmentation.

If you call `MoveHHi` to move a handle to a resource that has its `resChanged` bit set, the Resource Manager updates the resource by using the `WriteResource` function to write the contents of the block to disk. If you want to avoid this behavior, call the Resource Manager function `SetResPurge(FALSE)` before you call `MoveHHi`, and then call `SetResPurge(TRUE)` to restore the default setting.

By using the `MoveHHi` function on relocatable blocks you plan to allocate for short periods of time, you help prevent islands of immovable memory from accumulating in (and thus fragmenting) the heap.

Do not use the `MoveHHi` function to move blocks you plan to lock for long periods of time. The `MoveHHi` function moves such blocks to the top of the heap, perhaps preventing other blocks already at the top of the heap from moving down once they are unlocked. Instead, use the `ReserveMem` function before allocating such blocks, thus keeping them in the bottom partition of the heap, where they do not prevent relocatable blocks from moving.

If you frequently lock a block for short periods of time and find that calling `MoveHHi` each time slows down your application, you might consider leaving the block always locked and calling the `ReserveMem` function before allocating it.

Once you move a block to the top of the heap, be sure to lock it if you do not want the Memory Manager to move it back to the middle partition as soon as it can. (The `MoveHHi` function cannot move locked blocks; be sure to lock blocks after, not before, calling `MoveHHi`.)

Using the `MoveHHi` function without taking other precautionary measures to prevent heap fragmentation is useless, because even one small nonrelocatable or locked relocatable block in the middle of the heap might prevent `MoveHHi` from moving blocks to the top of the heap.

Call the function `MemError` (page 27) to get the result code. See "Memory Manager Result Codes" (page 47).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`MacMemory.h`

NewGrowZoneUPP

(Deprecated in Mac OS X v10.4. There is no replacement function; heaps never grow in Mac OS X, so the grow-zone function is never called.)

```
GrowZoneUPP NewGrowZoneUPP (
   GrowZoneProcPtr userRoutine
);
```

Parameters

userRoutine

Return Value
See the description of the `GrowZoneUPP` data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

NewPurgeUPP

(Deprecated in Mac OS X v10.4. There is no replacement function; heaps are never purged in Mac OS X, so the purge function is never called.)

```
PurgeUPP NewPurgeUPP (
   PurgeProcPtr userRoutine
);
```

Parameters

userRoutine

Return Value
See the description of the `PurgeUPP` data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

NewUserFnUPP

(Deprecated in Mac OS X v10.4. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
UserFnUPP NewUserFnUPP (
   UserFnProcPtr userRoutine
);
```

Parameters

userRoutine

Return Value
See the description of the `UserFnUPP` data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

PurgeMem

Purges the current heap zone until the specified number of bytes are available. (Deprecated in Mac OS X v10.4. There is no replacement; heaps are never purged in Mac OS X, so this function does nothing.)

```
void PurgeMem (
   Size cbNeeded
);
```

Parameters

cbNeeded

The size, in bytes, of the block for which `PurgeMem` should attempt to make room.

Discussion
The Memory Manager purges the heap automatically when a memory request fails. However, you can use `PurgeMem` to purge the current heap zone manually.

The `PurgeMem` function sequentially purges blocks from the current heap zone until it either allocates a contiguous block of the specified size or purges the entire zone. If `PurgeMem` purges the entire zone without creating a contiguous block of the specified size, `PurgeMem` generates the result code `memFullErr`.

Call the function `MemError` (page 27) to get the result code. See "Memory Manager Result Codes" (page 47).

The `PurgeMem` function purges only relocatable, unlocked, purgeable blocks. The function does not actually attempt to allocate the memory.

To purge the entire heap zone, call `PurgeMem(maxSize)`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

PurgeSpace

Determines the total amount of free memory and the size of the largest allocatable block in the current heap zone if it were purged. (Deprecated in Mac OS X v10.4. There is no replacement; heaps are never purged in Mac OS X.)

```
void PurgeSpace (
   long *total,
   long *contig
);
```

Parameters

total

On return, the total amount of free memory, in bytes, in the current heap zone if it were purged. This amount includes space that is already free.

contig

On return, the size of the largest contiguous block of free memory in the current heap zone if it were purged.

Discussion

The `PurgeSpace` function does not actually purge the current heap zone.

Call the function `MemError` (page 27) to get the result code. See "Memory Manager Result Codes" (page 47).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`MacMemory.h`

PurgeSpaceContiguous

(Deprecated in Mac OS X v10.4. There is no replacement; heaps are never purged in Mac OS X.)

```
long PurgeSpaceContiguous (
   void
);
```

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`MacMemory.h`

PurgeSpaceTotal

(Deprecated in Mac OS X v10.4. There is no replacement; heaps are never purged in Mac OS X.)

```
long PurgeSpaceTotal (
   void
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

ReleaseMemoryData

Releases the data of a portion of the address space. (Deprecated in Mac OS X v10.4. There is no replacement; this function does nothing in Mac OS X.)

```
OSErr ReleaseMemoryData (
   void *address,
   unsigned long count
);
```

Return Value
This function always returns a value of `noErr`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

ReserveMem

Reserves space for a block of memory as close to the bottom of the current heap zone as possible. (Deprecated in Mac OS X v10.4. There is no replacement; this function does nothing in Mac OS X.)

```
void ReserveMem (
   Size cbNeeded
);
```

Parameters
`cbNeeded`
The number of bytes to reserve near the bottom of the heap.

Discussion
The `ReserveMem` function attempts to create free space for the specified number of contiguous logical bytes at the lowest possible position in the current heap zone. It pursues every available means of placing the block as close as possible to the bottom of the zone, including moving other relocatable blocks upward, expanding the zone (if possible), and purging blocks from it.

Use the `ReserveMem` function when allocating a relocatable block that you intend to lock for long periods of time. This helps prevent heap fragmentation because it reserves space for the block as close to the bottom of the heap as possible. Consistent use of `ReserveMem` for this purpose ensures that all locked, relocatable blocks and nonrelocatable blocks are together at the bottom of the heap zone and thus do not prevent unlocked relocatable blocks from moving about the zone.

Because `ReserveMem` does not actually allocate the block, you must combine calls to `ReserveMem` with calls to the `NewHandle` function.

Do not use the `ReserveMem` function for a relocatable block you intend to lock for only a short period of time. If you do so and then allocate a nonrelocatable block above it, the relocatable block becomes trapped under the nonrelocatable block when you unlock that relocatable block.

It isn't necessary to call `ReserveMem` to reserve space for a nonrelocatable block, because the `NewPtr` function calls it automatically.

Also, you do not need to call `ReserveMem` to reserve memory before you load a locked resource into memory, because the Resource Manager calls `ReserveMem` automatically.

Call the function `MemError` (page 27) to get the result code. See "Memory Manager Result Codes" (page 47).

Because the `ReserveMem` function could move and purge memory, you should not call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

SetGrowZone

Specifies the current heap zone's grow-zone function. (Deprecated in Mac OS X v10.4. There is no replacement function; heaps never grow in Mac OS X, so the grow-zone function is never called.)

```
void SetGrowZone (
   GrowZoneUPP growZone
);
```

Parameters

growZone

A pointer to the grow-zone function. A `NULL` value removes any previous grow-zone function from the zone.

Discussion
To specify a grow-zone function for the current heap zone, pass a pointer to that function to the `SetGrowZone` function. Usually you call this function early in the execution of your application.

If you initialize your own heap zones besides the application and system zones, you can alternatively specify a grow-zone function as a parameter to the `InitZone` function.

The Memory Manager calls the grow-zone function only after exhausting all other avenues of satisfying a memory request, including compacting the zone, increasing its size (if it is the original application zone and is not yet at its maximum size), and purging blocks from it.

See "Grow-Zone Operations" for a complete description of a grow-zone function.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

TempFreeMem

Returns the maximum amount of free memory in the temporary heap. (Deprecated in Mac OS X v10.4. There is no replacement function; Mac OS X does not have a separate temporary memory heap.)

```
long TempFreeMem (
   void
);
```

Return Value
The total amount of free temporary memory, in bytes, that you could allocate by calling `TempNewHandle`. Because these bytes might be dispersed throughout memory, it is ordinarily not possible to allocate a single relocatable block of that size.

Discussion
Returns the total amount of memory available for temporary allocation.

Special Considerations

In Mac OS X, there is no separate temporary memory heap. This function always returns a large value, because virtual memory is always available to fulfill any request for memory. You can assume that any reasonable memory allocation request will succeed.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

TempHLock

Locks a relocatable block in the temporary heap. (Deprecated in Mac OS X v10.4. Use `HLock` (page 22) instead; Mac OS X does not have a separate temporary memory heap.)

```
void TempHLock (
   Handle h,
   OSErr *resultCode
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
MacMemory.h

TempHUnlock

Unlocks a relocatable block in the temporary heap. (Deprecated in Mac OS X v10.4. Use `HUnlock` (page 25) instead; Mac OS X does not have a separate temporary memory heap.)

```
void TempHUnlock (
   Handle h,
   OSErr *resultCode
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
MacMemory.h

TempMaxMem

Returns the maximum amount of temporary memory available. (Deprecated in Mac OS X v10.4. There is no replacement function; Mac OS X does not have a separate temporary memory heap.)

```
Size TempMaxMem (
   Size *grow
);
```

Parameters

grow

On return, this parameter always contains 0 after the function call because temporary memory does not come from the application's heap zone, and only that zone can grow. Ignore this parameter.

Return Value
The size of the largest contiguous block available for temporary allocation.

Discussion
Compacts the current heap zone and returns the size of the largest contiguous block available for temporary allocation.

Special Considerations

In Mac OS X, there is no separate temporary memory heap. This function always returns a large value, because virtual memory is always available to fulfill any request for memory. You can assume that any reasonable memory allocation request will succeed.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`MacMemory.h`

TempTopMem

Returns the location of the top of the temporary heap. (Deprecated in Mac OS X v10.4. There is no replacement function; Mac OS X does not have a separate temporary memory heap.)

```
Ptr TempTopMem (
   void
);
```

Discussion

In Mac OS X, this function always returns `NULL`.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`MacMemory.h`

TopMem

Returns a pointer to the byte at the top of an application's partition. (Deprecated in Mac OS X v10.4. There is no replacement; this function does nothing in Mac OS X.)

```
Ptr TopMem (
   void
);
```

Discussion

Deprecated. Refer to `MacMemory.h` for information on replacement functions.

`TopMem` obtains a pointer to the byte at the top of an application's partition, directly above the jump table. `TopMem` does this to maintain compatibility with programs that check `TopMem` to find out how much memory is installed in a computer. The preferred method of obtaining this information is with the `Gestalt` function.

The function exhibits special behavior at startup time, and the value it returns controls the amount by which an extension can lower the value of the global variable `BufPtr` at startup time. If you are writing a system extension, you should not lower the value of `BufPtr` by more than `MemTop DIV 2 + 1024`. If you do lower `BufPtr` too far, the startup process generates an out-of-memory system error.

You should never need to call `TopMem` except during the startup process.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

UnholdMemory

Makes a currently held range of memory eligible for paging again. (Deprecated in Mac OS X v10.4. There is no replacement; this function does nothing in Mac OS X.)

```
OSErr UnholdMemory (
   void *address,
   unsigned long count
);
```

Parameters

address
A pointer indicating the starting address of the range of memory to be released.

count
The size, in bytes, of the range of memory to be released.

Return Value
This function always returns a value of `noErr`.

Discussion
If the starting address you supply to the `UnholdMemory` function is not on a page boundary, then `UnholdMemory` rounds down to the nearest page boundary. Similarly, if the specified range does not end on a page boundary, `UnholdMemory` rounds up the value you pass in the `count` parameter so that the entire range of memory is released.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

Deprecated in Mac OS X v10.5

FreeMem

Returns the total amount of free space in the current heap zone. (Deprecated in Mac OS X v10.5. There is no replacement function; you can assume that any reasonable memory allocation will succeed.)

```
long FreeMem (
   void
);
```

Return Value
Returns a fixed value for heap size that is compatible with most applications.

Discussion
In Mac OS 8 and 9, this function returns the total amount of free space in the current heap zone. In Mac OS X, this function always returns a large fixed value because applications run in a large, protected memory space.

Call the function `MemError` (page 27) to get the result code. See "Memory Manager Result Codes" (page 47).

Special Considerations

Even though `FreeMem` does not move or purge memory, you should not call it at interrupt time because the heap might be in an inconsistent state.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

MaxBlock

Returns a fixed value for block size that is compatible with most applications. (Deprecated in Mac OS X v10.5. There is no replacement function; you can assume that any reasonable memory allocation will succeed.)

```
long MaxBlock (
   void
);
```

Return Value
The maximum contiguous space, in bytes, that you could obtain after compacting the current heap zone. `MaxBlock` does not actually do the compaction.

Discussion
In Mac OS X, this function always returns a large value because virtual memory is always available to fulfill any request for memory.

Call the function `MemError` (page 27) to get the result code. See "Memory Manager Result Codes" (page 47).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`MacMemory.h`

MaxMem

Returns the size, in bytes, of the largest contiguous free block in the current heap zone. (Deprecated in Mac OS X v10.5. There is no replacement function; you can assume that any reasonable memory allocation will succeed.)

```
Size MaxMem (
   Size *grow
);
```

Parameters

grow

On return, the maximum number of bytes by which the current heap zone can grow. After a call to `MaxApplZone`, `MaxMem` always sets this parameter to 0.

Return Value

The size, in bytes, of the largest contiguous free block in the zone after the compacting and purging.

Discussion

In Mac OS 8 and 9, the `MaxMem` function compacts the current heap zone and purges all relocatable, unlocked, and purgeable blocks from the zone. If the current zone is the original application zone, the `grow` parameter is set to the maximum number of bytes by which the zone can grow. For any other heap zone, `grow` is set to 0. `MaxMem` does not actually expand the zone or call the zone's grow-zone function.

In Mac OS X, the `MaxMem` function returns a large fixed value because applications run in a large, protected memory space.

Call the function `MemError` (page 27) to get the result code. See "Memory Manager Result Codes" (page 47).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`MacMemory.h`

StackSpace

Returns the amount of space between the bottom of the stack and the top of the application heap. (Deprecated in Mac OS X v10.5. There is no replacement; this function was included to facilitate porting legacy applications to Carbon, but it serves no useful purpose in Mac OS X.)

```
long StackSpace (
   void
);
```

Return Value

The current amount of stack space, in bytes, between the current stack pointer and the application heap.

Discussion
Usually you determine the maximum amount of stack space needed before you ship your application. Thus this function is generally useful only during debugging to determine how big to make the stack. However, if your application calls a recursive function that conceivably could call itself many times, that function should keep track of the stack space and take appropriate action if it becomes too low.

Call the function `MemError` (page 27) to get the result code. See "Memory Manager Result Codes" (page 47).

Special Considerations

StackSpace must not be called at interrupt time, as it may alter location MemErr.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`MacMemory.h`

TempDisposeHandle

Releases a relocatable block in the temporary heap. (Deprecated in Mac OS X v10.5. Use `DisposeHandle` (page 16) instead; Mac OS X does not have a separate temporary memory heap.)

```
void TempDisposeHandle (
   Handle h,
   OSErr *resultCode
);
```

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`MacMemory.h`

Document Revision History

This table describes the changes to *Memory Manager Reference*.

Date	Notes
2007-06-27	Added information in Result Codes chapter that the "memROZErr" (page 48) value is not relevant in Mac OS X. Clarified discussion for the SetHandleSize (page 35) function.
2006-07-24	Added information about deprecated functions.
2005-08-11	Updated the Introduction to reflect the use of Memory Manager in Mac OS X. Moved deprecated functions to a separate group. Revised the descriptions of several functions to reflect their behavior in Mac OS X. Added text to the callbacks, data types, and constants sections indicating that they are not functional or used in Mac OS X.
2003-02-01	Updated formatting and linking. Moved unsupported functions to Appendix A.

Document Revision History

Index

K

L

M

N

P

R

S

T

U

V

Z