Memory Management Utilities Reference

Carbon > Resource Management

2006-07-12

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

Simultaneously published in the United States and Canada.

Contents

Memory Management Utilities Reference

Framework: CoreServices/CoreServices.h
Declared in OSUtils.h

Overview

Applications can use the Memory Management Utilities to

- ensure that their callback routines, interrupt tasks, and stand-alone code could access application global variables or QuickDraw global variables
- add elements to and remove them from an operating-system queue
- ensure that they function properly in both 24- and 32-bit modes
- ensure that data or instructions in the microprocessor's internal caches remain consistent with data or instructions in RAM

While Carbon supports most of the Memory Management Utilities, there are changes to functions that assume a 68K runtime environment.

- Functions that flush caches on 68K processors (such as `FlushInstructionCache`, `FlushDataCache`, and `FlushCodeCacheRange`) are no longer supported.
- Functions such as `SetA5` or `SetCurrentA5` do nothing when running in Mac OS X. However, these functions should work normally when running in Mac OS 8 or 9.
- The functions `GetMMUMode` and `SwapMMUMode` are not supported because all PowerPC applications use 32-bit addressing, even if they are not Carbon-compliant.
- The `SysEnvirons` function is no longer supported since the Gestalt Manager can provide the same information. You should call the functions `FindFolder` and `Gestalt` instead.

For a list of unsupported functions, see the Carbon Specification.

Functions by Task

Determining the Measurement System

`IsMetric` (page 11)
Verifies whether the current script system is using the metric system or the English system of measurement.

Reading and Writing Location Data

`ReadLocation` (page 13)
Obtains information about a geographic location or time zone.

`WriteLocation` (page 27) Deprecated in Mac OS X v10.0
Changes the geographic location or time-zone information stored in extended parameter RAM. (Deprecated. There is no replacement because you cannot set this information in Mac OS X.)

Setting and Restoring the A5 Register

`SetA5` (page 29) Deprecated in Mac OS X v10.4
Sets the A5 register to the address specified. (Deprecated. There is no replacement because Mac OS X doesn't use the A5 variable.)

`SetCurrentA5` (page 30) Deprecated in Mac OS X v10.4
Sets the value in register A5 to the value of the low-memory global variable CurrentA5. (Deprecated. There is no replacement because Mac OS X doesn't use the A5 variable.)

Getting the User and Computer Name

`CSCopyMachineName` (page 7)
Returns a reference to the CFString that represents the computer name.

`CSCopyUserName` (page 8)
Returns a reference to the CFString that represents the user name.

Managing a Queue

`Enqueue` (page 10)
Adds elements directly to an operating-system queue or a queue that you create.

`Dequeue` (page 8)
Removes a queue element directly from an operating-system queue or from a queue that you have created.

`DTInstall` (page 28) Deprecated in Mac OS X v10.4
Adds the specified task record to the deferred-task queue. (Deprecated. You should restructure your application to use threads, such as those supplied by Multiprocessing Services.)

`DTUninstall` (page 29) Deprecated in Mac OS X v10.4
(Deprecated. You should restructure your application to use threads, such as those supplied by Multiprocessing Services.)

Working With Parameter RAM

`InitUtil` (page 28) Deprecated in Mac OS X v10.3
Copies the contents of parameter RAM into low memory. (Deprecated. There is no replacement because Mac OS X doesn't require this initialization.)

`GetSysPPtr` (page 29) Deprecated in Mac OS X v10.4
: Returns a pointer to the low-memory copy of parameter RAM. (Deprecated. There is no replacement; this function always returns `NULL` in Mac OS X.)

`WriteParam` (page 31) Deprecated in Mac OS X v10.4
: Write the modified values in the system parameters data structure to parameter RAM. (Deprecated. There is no replacement, because this function does nothing in Mac OS X.)

Miscellaneous

`Delay` (page 8)
: Delays execture for the specified amount of time.

`TickCount` (page 14)
: Obtains the current number of ticks (a tick is approximately 1/60 of a second) since the system last started up.

`MakeDataExecutable` (page 12)
: Notifies the system that the specified data is subject to execution.

Working With Universal Procedure Pointers

`NewDeferredTaskUPP` (page 13)
: Creates a new universal procedure pointer (UPP) to a deferred-task callback.

`InvokeDeferredTaskUPP` (page 11)
: Calls a deferred-task callback.

`DisposeDeferredTaskUPP` (page 10)
: Disposes of a universal procedure pointer (UPP) to a deferred-task callback.

Functions

CSCopyMachineName

Returns a reference to the CFString that represents the computer name.

```
CFStringRef CSCopyMachineName (
   void
);
```

Return Value
A `CFStringRef`. See the Base Services documentation for a description of the `CFStringRef` data type.

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

Declared In
`OSUtils.h`

CSCopyUserName

Returns a reference to the CFString that represents the user name.

```
CFStringRef CSCopyUserName (
   Boolean useShortName
);
```

Parameters

useShortName

A Boolean value that specifies whether to return the short name or full name of the user.

Return Value

A `CFStringRef`. See the Base Services documentation for a description of the `CFStringRef` data type.

Discussion

The function `CSCopyUserName` returns a `CFStringRef` based on the read UID (RUID, as returned by `getuid`) of the calling process. This can result in unexpected behavior (that is, `CSCopyUserName` returning different results than `SCDynamicStoreCopyConsoleUser`) for processes that manipulate their UID.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSUtils.h`

Delay

Delays execture for the specified amount of time.

```
void Delay (
   unsigned long numTicks,
   unsigned long *finalTicks
);
```

Parameters

numTicks

finalTicks

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSUtils.h`

Dequeue

Removes a queue element directly from an operating-system queue or from a queue that you have created.

```
OSErr Dequeue (
   QElemPtr qElement,
   QHdrPtr qHeader
);
```

Parameters

qElement

A pointer to a queue element to remove from a queue.

qHeader

A pointer to a queue header.

Return Value

A result code. See "Memory Management Utilities Result Codes" (page 24).

Discussion

The `Dequeue` function attempts to find the queue element specified by the `qElement` parameter in the queue specified by the `qHeader` parameter. If `Dequeue` finds the element, it removes the element from the queue, adjusts the other elements in the queue accordingly, and returns `noErr`. Otherwise, it returns `qErr`, indicating that it could not find the element in the queue. The `Dequeue` function does not deallocate the memory occupied by the queue element.

For a description of the `QElem` record, see `QElem` (page 17); for a description of the `QHdr` record, see `QHdr` (page 18).

The `Dequeue` function disables interrupts as it searches through the queue for the element to be removed. The time during which interrupts are disabled depends on the length of the queue and the position of the entry in the queue. The `Dequeue` function can be called at interrupt time. However, the `Dequeue` function is ordinarily used only by system software and, whenever possible, you should manipulate an operating-system queue indirectly, by calling special-purpose removal functions. You can use the Queue Utilities functions for directly manipulating queues that you create. Use the following functions instead of `Dequeue`:

- `SlotVRemove`

  Removes a slot-based VBL task. This function is available with the Vertical Retrace Manager.

- `VRemove`

  Removes a system-based VBL task. This function is available with the Vertical Retrace Manager.

- `WaitNextEvent`

  Removes an Event. This function is available with the Event manager.

- `SIntRemove`

  Removes a slot interrupt task. This function is available with the Slot Manager.

- `NMRemove`

  Removes a Notification request. This function is available with the Notification Manager.

- `SleepQRemove`

  Removes a Sleep task. This function is available with the Power Manager.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSUtils.h`

DisposeDeferredTaskUPP

Disposes of a universal procedure pointer (UPP) to a deferred-task callback.

```
void DisposeDeferredTaskUPP (
   DeferredTaskUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer.

Discussion

See the callback `DeferredTaskProcPtr` (page 15) for more information.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSUtils.h`

Enqueue

Adds elements directly to an operating-system queue or a queue that you create.

```
void Enqueue (
   QElemPtr qElement,
   QHdrPtr qHeader
);
```

Parameters

qElement

A pointer to the queue element to add to a queue.

qHeader

A pointer to a queue header.

Discussion

The `Enqueue` function adds the queue element specified by the `qElement` parameter to the end of the queue specified by the `qHeader` parameter. The specified queue header is updated to reflect the new queue element.

For a description of the `QElem` record, see `QElem` (page 17); for a description of the `QHdr` record, see `QHdr` (page 18).

Because interrupt functions are likely to manipulate operating-system queues, interrupts are disabled for a short time while the specified queue is updated. You can call the `Enqueue` function at interrupt time. However, the `Enqueue` function is ordinarily used only by system software. Whenever possible, you should manipulate an operating-system queue indirectly, by calling special-purpose functions whenever possible, instead of the `Enqueue` function. You can use the Queue Utilities functions for directly manipulating queues that you create. Use the following functions instead of `Enqueue`:

- `SlotVInstall`

  Installs a slot-based VBL task. This function is available with the Vertical Retrace Manager.

- `VInstall`

Installs a system-based VBL task. This function is available with the Vertical Retrace Manager.

- `AddDrive`

  Adds a disk drive. This function is available with the Device Manager.

- `PPostEvent` and `PostEvent`

  Installs an Event. This function is available with the Event manager.

- `DTInstall`

  Installs a deferred task. This function is available with the Memory Management Utilities.

- `SIntInstall`

  Installs a slot interrupt task. This function is available with the Slot Manager.

- `NMInstall`

  Installs a Notification request. This function is available with the Notification Manager.

- `SleepQInstall`

  Installs a Sleep task. This function is available with the Power Manager.

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

Declared In
`OSUtils.h`

InvokeDeferredTaskUPP

Calls a deferred-task callback.

```
void InvokeDeferredTaskUPP (
   long dtParam,
   DeferredTaskUPP userUPP
);
```

Discussion
You should not need to use the function `InvokeDeferredTaskUPP`, as the system calls your deferred-task callback for you. See the callback `DeferredTaskProcPtr` (page 15) for more information.

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

Declared In
`OSUtils.h`

IsMetric

Verifies whether the current script system is using the metric system or the English system of measurement.

```
Boolean IsMetric (
   void
);
```

Return Value
`TRUE` if the metric system is being used; `FALSE` if the English system is being used.

Discussion
The `IsMetric` function examines the `metricSys` field of the numeric-format resource (resource type `'itl0'`) to determine if the current script is using the metric system. A value of 255 in the `metricSys` field indicates that the metric system (centimeters, kilometers, milligrams, degrees Celsius, and so on) is being used. A value of 0 in the `metricSys` field indicates that the English system of measurement (inches, miles, ounces, degrees Fahrenheit, and so on) is used.

If you want to use units of measurement different from that of the current script, you need to override the value of the `metricSys` field in the current numeric-format resource. You can do this by using your own version of the numeric-format resource instead of the current script system's default international resource.

The `IsMetric` function is the same as the `IUMetric` function, which was previously available with the International Utilities Package.

Special Considerations

The `IsMetric` function may move or purge blocks in the heap calling it may cause problems if you've dereferenced a handle. Do not call this function from within interrupt code, such as in a completion function or a VBL task.

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

Declared In
`OSUtils.h`

MakeDataExecutable

Notifies the system that the specified data is subject to execution.

```
void MakeDataExecutable (
   void *baseAddress,
   unsigned long length
);
```

Parameters

`baseAddress`
The starting address of the data to be flushed.

`length`
The length of the data pointed to by the `baseAddress` parameter.

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

Declared In
`OSUtils.h`

NewDeferredTaskUPP

Creates a new universal procedure pointer (UPP) to a deferred-task callback.

```
DeferredTaskUPP NewDeferredTaskUPP (
   DeferredTaskProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your deferred-task callback.

Return Value

On return, a UPP to a deferred-task callback. See the description of the `DeferredTaskUPP` data type.

Discussion

See the callback `DeferredTaskProcPtr` (page 15) for more information.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSUtils.h`

ReadLocation

Obtains information about a geographic location or time zone.

```
void ReadLocation (
   MachineLocation *loc
);
```

Parameters

loc

On return, the fields of the geographic location structure containing the geographic location and the time-zone information.The `ReadLocation` procedure reads the stored geographic location and time zone of the Macintosh computer from extended parameter RAM.

You can get values for the latitude, longitude, daylight savings time (DST), or Greenwich mean time (GMT). If the geographic location record has never been set, all fields contain 0.

Discussion

The latitude and longitude are stored as `Fract` values, giving accuracy to within one foot. For example, a `Fract` value of 1.0 equals 90 degrees –1.0 equals –90 degrees and –2.0 equals –180 degrees.

To convert these values to a degrees format, you need to convert the `Fract` values first to the `Fixed` data type, then to the `LongInt` data type. Use the Mathematical and Logical Utilities functions `Fract2Fix` and `Fix2Long` to accomplish this task.

The DST value is a signed byte value that specifies the offset for the `hour` field—whether to add one hour, subtract one hour, or make no change at all.

The GMT value is in seconds east of GMT. For example, San Francisco is at –28,800 seconds (8 hours * 3,600 seconds per hour) east of GMT. The `gmtDelta` field is a 3-byte value contained in a long word, so you must take care to get it properly.

The `ReadLocation` function was previously available with the Script Manager.

For more information on the geographic location record, see `MachineLocation` (page 16).

For more information on the `Fract` data type and the conversion routines `Long2Fix`, `Fix2Fract`, `Fract2Fix`, and `Fix2Long`, see Mathematical and Logical Utilities.

Special Considerations

Do not call the `ReadLocation` function at interrupt time.

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

Declared In
`OSUtils.h`

TickCount

Obtains the current number of ticks (a tick is approximately 1/60 of a second) since the system last started up.

```
UInt32 TickCount (
   void
);
```

Discussion
The `TickCount` function returns an unsigned 32-bit integer that indicates the current number of ticks since the system last started up. You can use this value to compare the number of ticks that have elapsed since a given event or other action occurred. For example, you could compare the current value returned by `TickCount` with the value of the `when` field of an event structure.

The tick count is incremented during the vertical retrace interrupt, but this interrupt can be disabled. Your application should not rely on the tick count to increment with absolute precision. Your application also should not assume that the tick count always increments by 1 an interrupt task might keep control for more than one tick. If your application keeps track of the previous tick count and then compares this value with the current tick count, your application should compare the two values by checking for a “greater than or equal” condition rather than “equal to previous tick count plus 1.”

Do not rely on the tick count being exact; it is usually accurate to within one tick, but this level of accuracy is not guaranteed.

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

Related Sample Code
QTCarbonShell
Simple DrawSprocket

Declared In
`OSUtils.h`

Callbacks

DeferredTaskProcPtr

Defines a pointer to a deferred-task callback.

```
typedef void (*DeferredTaskProcPtr) (
    long dtParam
);
```

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

```
void MyDeferredTaskProc (
    long dtParam
);
```

Parameters

dtParam

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSUtils.h`

Data Types

DeferredTask

Contains information related to a deferred task.

```
struct DeferredTask {
    volatile QElemPtr qLink;
    short qType;
    volatile short dtFlags;
    DeferredTaskUPP dtAddr;
    long dtParam;
    long dtReserved;
};
typedef struct DeferredTask DeferredTask;
typedef DeferredTask * DeferredTaskPtr;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSUtils.h`

DeferredTaskUPP

Defines a universal procedure pointer to a deferred-task callback.

```
typedef DeferredTaskProcPtr DeferredTaskUPP;
```

Discussion
For more information, see the description of the `DeferredTaskProcPtr` (page 15) callback function.

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

Declared In
`OSUtils.h`

MachineLocation

Contains information about the geographical location of a computer.

```
struct MachineLocation {
    Fract latitude
    Fract longitude
    union {
    #if TARGET_RT_BIG_ENDIAN
        SInt8 dlsDelta;
    #endif
        long gmDelta;
        struct {
        #if TARGET_RT_LITTLE_ENDIAN
            SInt8 pad[3];
        #endif
            SInt8 Delta;
        } dls;
    } u;
};
typedef struct MachineLocation MachineLocation;
```

Fields

`latitude`
The location's latitude, in fractions of a great circle. For example, Copenhagen, Denmark is at 55.43 degrees north latitude. When writing the latitude to extended parameter RAM with the `WriteLocation` procedure, you must convert this value to a `Fract` data type. (For example, a `Fract` value of 1.0 equals 90 degrees –1.0 equals –90 degrees and –2.0 equals –180 degrees.) For more information on the `Fract` data type, see Mathematical and Logical Utilities.

`longitude`
The location's longitude, in fractions of a great circle. For example, Copenhagen, Denmark is at 12.34 degrees east longitude. When writing the longitude to extended parameter RAM with the `WriteLocation` procedure, you must convert this value to a `Fract` data type. (For example, a `Fract` value of 1.0 equals 90 degrees –1.0 equals –90 degrees and –2.0 equals –180 degrees.

`dlsDelta`
A value that represents the current state of daylight savings time.

`gmDelta`
`pad`
`delta`

A signed byte value representing the hour offset for daylight saving time. This field is a 1-byte value contained in a long word. It should be preserved when writing `gmtDelta`.

Discussion
The geographic location and time-zone information of a Macintosh computer are stored in extended parameter RAM. The `MachineLocation` data type defines the format for the geographic location record.

The `ReadLocation` and `WriteLocation` procedures use the geographic location record to read and store the geographic location and time zone information in extended parameter RAM. If the geographic location record has never been set, all fields contain 0.

In order for MachineLocation to be endian-safe, a new member has been added to the 'u' union in the structure. You are encouraged to use the new member instead of the old one.

If your code looked like this:

```
MachineLocation.u.dlsDelta = 1;
```

you should change it to this:

```
MachineLocation.u.dls.Delta = 1;
```

to be endian safe. The `gmtDelta` remains the same; the low 24-bits are used. Remember that order of assignment DOES matter.

This will overwrite results:

```
MachineLocation.u.dls.Delta = 0xAA;          // u = 0xAAGGGGGG; G=Garbage
MachineLocation.u.gmtDelta = 0xBBBBBB;       // u = 0x00BBBBBB;
```

when in fact reversing the assignment would have preserved the values:

```
MachineLocation.u.gmtDelta = 0xBBBBBB;       // u = 0x00BBBBB;
MachineLocation.u.dls.Delta = 0xAA;          // u = 0xAABBBBBB;
```

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

Declared In
`OSUtils.h`

QElem

Contains information about a queue element.

```
struct QElem {
    QElem * qLink;
    short qType;
    short qData[1];
};
typedef struct QElem QElem;
typedef QElem * QElemPtr;
```

Fields

`qLink`

The type of the queue element. For a description of the values which you can use in this field, see "Queue Types" (page 22).

`qType`

A variable array of data. The type of data and the length depend upon the queue type, specified in the `qType` field.

`qData`

Discussion

A queue element is a single entry in a queue. Each operating-system queue created and maintained by the Macintosh Operating System consists of a queue header and a linked list of queue elements. The exact structure of an element in an operating-system queue depends on the type of the queue. The `QElem` data type defines the available queue elements.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSUtils.h`

QHdr

Contains information about the event queue.

```
struct QHdr {
    volatile short qFlags;
    volatile QElemPtr qHead;
    volatile QElemPtr qTail;
};
typedef struct QHdr QHdr;
typedef QHdr * QHdrPtr;
```

Fields

`qFlags`

Queue flags.

`qHead`

First queue entry.

`qTail`

Last queue entry.

Discussion

The event queue consists of a header followed by the actual entries in the queue. The event queue has the same header as all standard Macintosh Operating System queues. The `Qhdr` structure defines the queue header.

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

Declared In
`OSUtils.h`

SysEnvRec

Contains information about the system environment.

```
struct SysEnvRec {
    short environsVersion;
    short machineType;
    short systemVersion;
    short processor;
    Boolean hasFPU;
    Boolean hasColorQD;
    short keyBoardType;
    short atDrvrVersNum;
    short sysVRefNum;
};
typedef struct SysEnvRec SysEnvRec;
```

Fields

`environsVersion`

The version number of the `SysEnvirons` function that was used to fill in the record.

When you call the `SysEnvirons` function, you specify a version number to ensure that you receive a system environment record that matches your expectations. If you request a more recent version of `SysEnvirons` than is available, `SysEnvirons` places its own version number in the `environsVersion` field and returns a function result `envVersTooBig`.

`machineType`

A code for the Macintosh model. See "Macintosh Model Codes" (page 21). Use the `Gestalt` function to obtain information about machine types not listed among these constants.

`systemVersion`

The version number of the current System file, represented as two byte-long numbers with one or more implied decimal points. The value $0410, for example, represents system software version 4.1.

If you call `SysEnvirons` when a system earlier than 4.1 is running, the MPW glue places $0 in this field and returns a result code of `envNotPresent`.

`processor`

A code for the microprocessor. See "Microprocessor Codes" (page 22).

`hasFPU`

A Boolean value that indicates whether hardware floating-point processing is available.

`hasColorQD`

A Boolean value that indicates whether Color QuickDraw is present. This field says nothing about the presence of a color monitor.

`keyBoardType`

A code for the keyboard type. See "Keyboard Constants" (page 21). Use the `Gestalt` function to obtain information about keyboard types not listed among these constants.

If the Apple Desktop Bus is in use, this field returns the keyboard type of the keyboard on which the last keystroke was made.

`atDrvrVersNum`
: The version number of the AppleTalk driver (specifically, the .MPP driver) currently installed. If AppleTalk is not loaded, this field is 0.

`sysVRefNum`
: The working-directory reference number of the folder or volume that holds the open System file.

Discussion
The `SysEnvirons` function fills in a system environment record, which describes some aspects of the software and hardware environment.

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

Declared In
`OSUtils.h`

SysParmType

Contains settings used by the system at startup.

```
struct SysParmType {
    UInt8 valid;
    UInt8 aTalkA;
    UInt8 aTalkB;
    UInt8 config;
    short portA;
    short portB;
    long alarm;
    short font;
    short kbdPrint;
    short volClik;
    short misc;
};
typedef struct SysParmType SysParmType;
typedef SysParmType * SysPPtr;
```

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

Declared In
`OSUtils.h`

Constants

Addressing Errors

Specify a type of addressing error.

```
enum {
    false32b = 0,
    true32b = 1
};
```

Constants

`false32b`

Indicates a 24-bit addressing error.

Available in Mac OS X v10.0 and later.

Declared in `OSUtils.h`.

`true32b`

Indicates a 32-bit addressing error.

Available in Mac OS X v10.0 and later.

Declared in `OSUtils.h`.

Keyboard Constants

Specify keyboard types.

```
enum {
    envUnknownKbd = 0,
    envMacKbd = 1,
    envMacAndPad = 2,
    envMacPlusKbd = 3,
    envAExtendKbd = 4,
    envStandADBKbd = 5,
    envPrtblADBKbd = 6,
    envPrtblISOKbd = 7,
    envStdISOADBKbd = 8,
    envExtISOADBKbd = 9
};
```

Macintosh Model Codes

Specify models of Macintosh computers.

```
enum {
    envMac = -1,
    envXL = -2,
    envMachUnknown = 0,
    env512KE = 1,
    envMacPlus = 2,
    envSE = 3,
    envMacII = 4,
    envMacIIx = 5,
    envMacIIcx = 6,
    envSE30 = 7,
    envPortable = 8,
    envMacIIci = 9,
    envMacIIfx = 11
};
```

Microprocessor Codes

Specify types of microprocessors.

```
enum {
    envCPUUnknown = 0,
    env68000 = 1,
    env68010 = 2,
    env68020 = 3,
    env68030 = 4,
    env68040 = 5
};
```

Queue Types

Specifies queue types.

```
enum {
    dummyType = 0,
    vType = 1,
    ioQType = 2,
    drvQType = 3,
    evType = 4,
    fsQType = 5,
    sIQType = 6,
    dtQType = 7,
    nmType = 8
};
typedef SignedByte QTypes;
```

Constants

`dummyType`

Reserved.

Available in Mac OS X v10.0 and later.

Declared in `OSUtils.h`.

`vType`
Specifies a vertical retrace queue type. See the Vertical Retrace Manager for more information.
Available in Mac OS X v10.0 and later.
Declared in `OSUtils.h`.

`ioQType`
Specifies a file I/O or driver I/O queue type.
Available in Mac OS X v10.0 and later.
Declared in `OSUtils.h`.

`drvQType`
Specifies a drive queue type.
Available in Mac OS X v10.0 and later.
Declared in `OSUtils.h`.

`evType`
Specifies an event queue type. See the Event Manager for more information.
Available in Mac OS X v10.0 and later.
Declared in `OSUtils.h`.

`fsQType`
Specifies a volume-control-block queue type.
Available in Mac OS X v10.0 and later.
Declared in `OSUtils.h`.

`sIQType`
Specifies a slot interrupt queue type. See the Slot Manager for more information.
Available in Mac OS X v10.0 and later.
Declared in `OSUtils.h`.

`dtQType`
Specifies a deferred task queue type. See Memory Management Utilities for more information.
Available in Mac OS X v10.0 and later.
Declared in `OSUtils.h`.

`nmType`
Specifies a notification queue type. See the Notification Manager for more information.
Available in Mac OS X v10.0 and later.
Declared in `OSUtils.h`.

Discussion
The different queue types that are accessible to your application are defined by the `QTypes` data type. Each of these enumerated queue types determines a different type of queue element. These constants are used in the `qtype` field of the `QElem` (page 17) structure.

Sorting Constants

Specify the result types for the function `RelString`.

```
enum {
    sortsBefore = -1,
    sortsEqual = 0,
    sortsAfter = 1
};
```

Constants

`sortsBefore`

Indicates the first string is less than the second string.

Available in Mac OS X v10.0 and later.

Declared in `OSUtils.h`.

`sortsEqual`

Indicates the first string is equivalent to the second string.

Available in Mac OS X v10.0 and later.

Declared in `OSUtils.h`.

`sortsAfter`

Indicates the first string is greater than the second string.

Available in Mac OS X v10.0 and later.

Declared in `OSUtils.h`.

Assorted Use Constants

Defines constants to indicate use of various things, such as to use MIDE or AppleTalk.

```
enum {
    useFree = 0,
    useATalk = 1,
    useAsync = 2,
    useExtClk = 3,
    useMIDI = 4
};
```

Version Number

Specifies the version of the current system environment.

```
enum {
    curSysEnvVers = 2
};
```

Result Codes

The most common result codes returned by Memory Management Utilities are listed in the table below. Memory Management Utilities may also return the following errors:

`noErr` (0)

`paramErr` (-50)

prWrErr (-87)
prInitErr (-88)
memROZErr (-99)
memFullErr (-108)
nilHandleErr (-109)
memWZErr (-111)
memPurErr (-112)
memBCErr (-115)
memLockedErr (-117)
notEnoughMemoryErr (-620)
notHeldErr (-621)
cannotMakeContiguousErr (-622)
notLockedErr (-623)
interruptsMaskedErr (-624)
cannotDeferErr (-625)

Result Code	Value	Description
qErr	-1	Queue element not found during deletion. Available in Mac OS X v10.0 and later.
vTypErr	-2	Invalid queue element. Available in Mac OS X v10.0 and later.
corErr	-3	Core routine number out of range Available in Mac OS X v10.0 and later.
unimpErr	-4	Unimplemented core routine. Available in Mac OS X v10.0 and later.
SlpTypeErr	-5	Invalid queue element. Available in Mac OS X v10.0 and later.
hwParamErr	-502	Processor does not support flushing a range. Available in Mac OS X v10.0 and later.

Deprecated Memory Management Utilities Functions

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

Deprecated in Mac OS X v10.0

WriteLocation

Changes the geographic location or time-zone information stored in extended parameter RAM. (Deprecated in Mac OS X v10.0. There is no replacement because you cannot set this information in Mac OS X.)

```
void WriteLocation (
   const MachineLocation *loc
);
```

Parameters

loc

The geographic location and time-zone information to write to the extended parameter RAM.

Discussion

The latitude and longitude are stored in the geographic location structure as `Fract` values, giving accuracy to within 1 foot. For example, a `Fract` value of 1.0 equals 90 degrees –1.0 equals –90 degrees and –2.0 equals –180 degrees.

Use the functions `Long2Fix` and `Fix2Fract` to convert longitude and latitude values to the `Fixed` data type and then to the `Fract` data type for storage.

Use the daylight savings time value signed byte value to specify the offset for the `hour` field—whether to add one hour, subtract one hour, or make no change at all.

The Greenwich mean time value is in seconds east of GMT. For example, San Francisco is at –28,800 seconds (8 hours * 3,600 seconds per hour) east of GMT.The `gmtDelta` field is a 3-byte value contained in a long word. When writing `gmtDelta`, mask off the top byte because it is reserved. Preserve the value of `dlsDelta`.

The `WriteLocation` function was previously available with the Script Manager.

For more information on the geographic location record, see `MachineLocation` (page 16).

For more information on the `Fract` data type and the conversion routines `Long2Fix`, `Fix2Fract`, `Fract2Fix`, and `Fix2Long`, see Mathematical and Logical Utilities.

Special Considerations

Do not call the `WriteLocation` function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.0.
Not available to 64-bit applications.

Declared In
OSUtils.h

Deprecated in Mac OS X v10.3

InitUtil

Copies the contents of parameter RAM into low memory. (Deprecated in Mac OS X v10.3. There is no replacement because Mac OS X doesn't require this initialization.)

```
OSErr InitUtil (
   void
);
```

Return Value
A result code. See "Memory Management Utilities Result Codes" (page 24).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.3.
Not available to 64-bit applications.

Declared In
OSUtils.h

Deprecated in Mac OS X v10.4

DTInstall

Adds the specified task record to the deferred-task queue. (Deprecated in Mac OS X v10.4. You should restructure your application to use threads, such as those supplied by Multiprocessing Services.)

```
OSErr DTInstall (
   DeferredTaskPtr dtTaskPtr
);
```

Parameters
dtTaskPtr

Return Value
A result code. See "Memory Management Utilities Result Codes" (page 24).

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
OSUtils.h

DTUninstall

(Deprecated in Mac OS X v10.4. You should restructure your application to use threads, such as those supplied by Multiprocessing Services.)

```
OSErr DTUninstall (
   DeferredTaskPtr dtTaskPtr
);
```

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

Declared In
OSUtils.h

GetSysPPtr

Returns a pointer to the low-memory copy of parameter RAM. (Deprecated in Mac OS X v10.4. There is no replacement; this function always returns `NULL` in Mac OS X.)

```
SysPPtr GetSysPPtr (
   void
);
```

Return Value
See the description of the `SysPPtr` 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
OSUtils.h

SetA5

Sets the A5 register to the address specified. (Deprecated in Mac OS X v10.4. There is no replacement because Mac OS X doesn't use the A5 variable.)

```
long SetA5 (
   long newA5
);
```

Parameters

newA5

The value to which the A5 register is to be changed.

Return Value
The value in the A5 register before SetA5 changes it to `newA5`.

Discussion
In interrupt code that accesses application global variables, use the `SetA5` function first to restore a value previously saved using `SetCurrentA5`, and then, at the end of the code, to restore the A5 register to the value it had before the first call to `SetA5`.

Carbon Porting Notes

68K-specific. Does nothing in PowerPC native code.

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
`OSUtils.h`

SetCurrentA5

Sets the value in register A5 to the value of the low-memory global variable CurrentA5. (Deprecated in Mac OS X v10.4. There is no replacement because Mac OS X doesn't use the A5 variable.)

```
long SetCurrentA5 (
   void
);
```

Return Value
The value in the A5 register before SetCurrentA5 changes it to the value of the low-memory global variable CurrentA5.

Discussion
The `CurrentA5` variable points to the boundary between the parameters and global variables of the current application.

You cannot reliably call `SetCurrentA5` in code that executes at interrupt time unless you first guarantee that your application is the current process (for example, by calling the Process Manager function `GetCurrentProcess`). In general, you should call `SetCurrentA5` at noninterrupt time and then pass the returned value to the interrupt code.

Carbon Porting Notes

68K-specific. Does nothing in PowerPC native code.

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
`OSUtils.h`

WriteParam

Write the modified values in the system parameters data structure to parameter RAM. (Deprecated in Mac OS X v10.4. There is no replacement, because this function does nothing in Mac OS X.)

```
OSErr WriteParam (
   void
);
```

Return Value
A result code. See "Memory Management Utilities Result Codes" (page 24).

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
`OSUtils.h`

Document Revision History

This table describes the changes to *Memory Management Utilities Reference*.

Date	Notes
2006-07-12	Made minor formatting changes.
2006-07-24	Added deprecation information.
2003-04-01	Added abstracts for 11 functions and several data types. Moved some functions from the Miscellaneous section to more meaningful categories.
2003-02-01	Updated formatting.
	Created a Working With Universal Procedure Pointers section.
2001-07-01	Last version of this document.

Document Revision History

Index

Q

R

S

T

U

V

W