inb(), inw(), inl(), outb(), outw(), outl()

SUMMARY Read or write data to an I/O port

DECLARED IN driverkit/i386/ioPorts.h

SYNOPSIS unsigned char **inb(**unsigned int *address***)**
unsigned short **inw(**unsigned int *address***)**
unsigned long **inl(**unsigned int *address***)**
void **outb(**unsigned int *address*, unsigned char *data***)**
void **outw(**unsigned int *address*, unsigned short *data***)**
void **outl(**unsigned int *address*, unsigned long *data***)**

DESCRIPTION These inline functions let drivers read and write I/O ports on Intel-based computers. Use **inb()** to read a byte at the I/O port *address*. Use **inw()** to read the two bytes at *address* and *address* + 1, and **inl()** to read four bytes starting at *address*. To write a byte, use **outb()**; to write two bytes (to *address* and *address* + 1), use **outw()**; to write four bytes, use **outl()**.

These functions have nothing to do with main memory; they work only for the 64 kilobytes of I/O address space on an Intel-based computer. These functions use the special machine instructions that are necessary for reading and writing data from and to the I/O space.

Note: These functions work only at kernel level and only on Intel-based computers.

EXAMPLE
```
temp_cr = inb(base+CR); /* get current CR value */
```

IOAddDDMEntry()

SUMMARY Add one entry to the Driver Debugging Module

DECLARED IN driverkit/debugging.h

SYNOPSIS void **IOAddDDMEntry(**char **format*, int *arg1*, int *arg2*, int *arg3*, int *arg4*, int *arg5***)**

DESCRIPTION This is the exported function that is used to add events to the DDM's circular buffer. However, drivers typically don't use this directly; instead, they should use macros that call **IOAddDDMEntry()** conditionally based on the current state of debugging flags. See the description of **IODEBUG()** for examples.

Note: The last 5 arguments to this function are typed above as **int**, but they are really untyped and could be any 32-bit quantity. They are stored in the debugging log as **int** but are eventually evaluated as arguments to **sprintf()**, so they could be **int**, **char**, **short**, or pointers to a string. See **IOCopyString()**, later in this section, for information on passing string pointers to **IOAddDDMEntry()**.

SEE ALSO **IODEBUG()**

IOAddToBdevsw(), IOAddToCdevsw(), IOAddToVfssw()

SUMMARY Add UNIX-style entry points to a device switch table

DECLARED IN driverkit/devsw.h

SYNOPSIS int **IOAddToBdevsw(**IOSwitchFunc *openFunc*, IOSwitchFunc *closeFunc*, IOSwitchFunc *strategyFunc*, IOSwitchFunc *dumpFunc*, IOSwitchFunc *psizeFunc*, BOOL *isTape***)**
int **IOAddToCdevsw(**IOSwitchFunc *openFunc*, IOSwitchFunc *closeFunc*, IOSwitchFunc *readFunc*, IOSwitchFunc *writeFunc*, IOSwitchFunc *ioctlFunc*, IOSwitchFunc *stopFunc*, IOSwitchFunc *resetFunc*, IOSwitchFunc *selectFunc*, IOSwitchFunc *mmapFunc*, IOSwitchFunc *getcFunc*, IOSwitchFunc *putcFunc***)**
int **IOAddToVfssw(**const char **vfsswName*, const struct vfsops **vfsswOps***)**

DESCRIPTION These functions find a free row in a device switch table and add the specified entry points. Each function returns the major number (equivalent to the row number) for the device, or -1 if the device couldn't be added to the table.

Note: You should use IODevice's **addToBdevsw...** and **addToCdevsw...** methods instead of **IOAddToBdevsw()** and **IOAddToCdevsw()**, whenever possible.

SEE ALSO **IORemoveFromBdevsw()**, **IORemoveFromCdevsw()**, **IORemoveFromVfssw()**

IOAlign()

SUMMARY Truncate an address so that it's aligned to a buffer size

DECLARED IN driverkit/align.h

SYNOPSIS *type* **IOAlign(***type*, *address*, *bufferSize***)**

DESCRIPTION This macro truncates *address* to a multiple of *bufferSize*.

SEE ALSO **IOIsAligned()**

IOClearDDM()

SUMMARY Clear the Driver Debugging Module's entries

DECLARED IN driverkit/debugging.h

SYNOPSIS void **IOClearDDM()**

DESCRIPTION This function empties the DDM's circular buffer.

IOConvertPort()

SUMMARY Convert a port name from one IPC space to another

DECLARED IN driverkit/kernelDriver.h

SYNOPSIS port_t **IOConvertPort(**port_t *port*, IOIPCSpace *from*, IOIPCSpace *to***)**

DESCRIPTION This function lets a kernel driver convert a port name (*port*) so that the port can be used

in a different IPC space. Three types of conversion are supported:

- From the current task's IPC space to the kernel I/O task's space
- From the kernel's IPC space to the kernel I/O task's space
- From the kernel I/O task's IPC space to kernel's IPC space

The arguments *from* and *to* should each be specified as one of the following: IO_Kernel, IO_KernelIOTask, or IO_CurrentTask. For example, the following code converts a port name from the current task's name to the name used by the kernel I/O task.

```
ioTaskPort = IOConvertPort(aPort, IO_CurrentTask, IO_KernelIOTask);
```

Note: This function works only in kernel-level drivers.

RETURN Returns the port's name in the *to* space. Specifying an invalid conversion results in a return value of PORT_NULL.

IOCopyMemory()

SUMMARY Copy memory using the specified transfer width

DECLARED IN driverkit/generalFuncs.h

SYNOPSIS void **IOCopyMemory(**void **from*, void **to*, unsigned int *numBytes*, unsigned int *bytesPerTransfer***)**

DESCRIPTION Copies memory 1, 2, or 4 bytes at a time (as specified by *bytesPerTransfer*) until *numBytes* bytes starting at *from* have been copied to *to*. The *from* and *to* buffers must not overlap.

This function is useful when devices have mapped memory that can be accessed in only 8-bit or 16-bit quantities. In these situations, **bcopy()** isn't appropriate, since it assumes 32-bit access to all memory involved.

If *from* is not aligned on a *bytesPerTransfer* boundary, **IOCopyMemory()** performs 8-bit transfers until it has reached a *bytesPerTransfer* boundary. Similarly, if the end of the *from* buffer extends past a *bytesPerTransfer* boundary, the remaining memory is copied 8 bits at a time.

IOCopyString()

SUMMARY Return a copy of the specified string

DECLARED IN driverkit/debugging.h

SYNOPSIS const char ***IOCopyString(**const char **instring***)**

DESCRIPTION This function is required when you want to use a pointer to a string whose existence is transitory as an argument. The reason for this is that the string won't be read until the Driver Debugging Module's buffer is examined, which could be a long time (minutes or more) after the call to **IOAddDDMEntry()**. By then, the string pointer passed to **IOAddDDMEntry()** no longer might no longer point to a useful string.

Warning: The string returned by this function is created with **IOMalloc()** and is never freed. Use this function with discretion.

IODEBUG()

SUMMARY Conditionally add one entry to the Driver Debugging Module

DECLARED IN driverkit/debugging.h

SYNOPSIS void **IODEBUG(**int *index*, int *mask*, char **format*, int *arg1*, int *arg2*, int *arg3*, int *arg4*, int *arg5***)**

DESCRIPTION This macro is used to add entries to the DDM's circular buffer. The entry is added only if both of the following are true:

- The C preprocessor flag DDM_DEBUG is defined.
- A bitwise and operation performed on *mask* and **IODDMMasks[***index***]** results in a nonzero result.

IODEBUG() is typically used to define other macros specific to a driver, as shown in the following example.

EXAMPLE

```
#define MY_INDEX      0

#define MY_INPUT       0x00000001    //
#define MY_OUTPUT      0x00000002    //
#define MY_OTHER       0x00000004    //

    #define logInput(x, a, b, c, d, e) \
        IODEBUG(MY_INDEX, MY_INPUT, x, a, b, c, d, e)

    #define logOutput(x, a, b, c, d, e) \
        IODEBUG(MY_INDEX, MY_OUTPUT, x, a, b, c, d, e)

    #define logOther(x, a, b, c, d, e) \
        IODEBUG(MY_INDEX, MY_OTHER, x, a, b, c, d, e)

    . . .
    IODDMMasks[MY_INDEX] = MY_INPUT | MY_OUTPUT;
    . . .
    logInput("Input error %d: %s\n", error, IOFindNameForValue(error,
        &errorList));
```

IODelay()

SUMMARY Wait (without blocking) for the indicated number of microseconds

DECLARED IN driverkit/generalFuncs.h

SYNOPSIS void **IODelay(**unsigned int *microseconds***)**

DESCRIPTION This is a quick, nonblocking version of **IOSleep()**.

Note: This function guarantees a *minimum* ªspinº delay in the user-level version; due to thread scheduling, the call to **IODelay()** could take much longer than the indicated time. This should not be a problem with properly designed user-level drivers as this is a common real-time constraint on all user-level code.

IODisableInterrupt()

SUMMARY Prevent interrupt messages from being sent

DECLARED IN driverkit/IODirectDevice.h

SYNOPSIS void **IODisableInterrupt(**void ***identity***)

DESCRIPTION This function allows handlers of non-shared interrupts to indicate that the interrupt should be left disabled on return from the interrupt handler.

The *identity* argument should be set to the value that the interrupt handler received in its own arguments.

Note: **IODisableInterrupt()** must be called inside a special interrupt handler function. It can't be called from any other context.

SEE ALSO **IOEnableInterrupt(), IOSendInterrupt()**

IOEnableInterrupt()

SUMMARY Allow interrupt messages to be sent

DECLARED IN driverkit/IODirectDevice.h

SYNOPSIS void **IOEnableInterrupt(**void ***identity***)

DESCRIPTION This function allows interrupt handlers to indicate that the interrupt should be reenabled on return from the interrupt handler. You should only re-enable the interrupt after removing the source of the interruptÐby clearing the interrupt status register on the device, or by using whatever mechanism is necessary for the hardware your driver controls.

The *identity* argument should be set to the value that the interrupt handler received in its own arguments.

Note: **IOEnableInterrupt()** must be called inside a special interrupt handler function. It can't be called from any other context.

SEE ALSO **IODisableInterrupt(), IOSendInterrupt()**

IOExitThread()

SUMMARY Terminate the execution of the current thread

DECLARED IN driverkit/generalFuncs.h

SYNOPSIS volatile void **IOExitThread()**

DESCRIPTION This function terminates the execution of the current (calling) thread. Note that there's no way for one thread to kill another thread other than by sending some kind of message to the soon-to-be-terminated thread instructing it to kill itself.

Note: In the user-level implementation, the main C thread (the first thread in the task) doesn't exit until all other C threads in the task have exited.

IOFindNameForValue(), IOFindValueForName()

SUMMARY Convert an integer to a string, or vice versa, using an **IONamedValues** array

DECLARED IN driverkit/generalFuncs.h

SYNOPSIS const char ***IOFindNameForValue(**int *value*, const IONamedValues **array***)*
IOReturn **IOFindValueForName(**const char **string*, const IONamedValue **array*, int **value***)*

DESCRIPTION These functions are the primary use of the **IONamedValues** data type, which maps integer values to strings. **IOFindNameForValue()** maps a given integer value to a string, given a pointer to an array of **IONamedValues**. **IOFindValueForName()** maps a given string into an integer, returning the integer in *value*.

One typical use for **IOFindNameForValue()** is to map integer return values into error strings. IODevice's **IOStringFromReturn:** method performs this function. A subclass that defines additional IOReturn values should override this method and call **[super IOReturnToString:]** if the specified value does not match one of the class-specific IOReturns.

RETURN **IOFindNameForValue()** returns the string corresponding to *value*, or a string indicating that *value* is undefined if the integer wasn't found. **IOFindValueForName()** returns IO_R_SUCCESS if it finds the specified string; otherwise, it returns IO_R_INVALIDARG.

IOForkThread()

SUMMARY Start a new thread

DECLARED IN driverkit/generalFuncs.h

SYNOPSIS IOThread **IOForkThread(**IOThreadFunc *function*, void **arg***)*

DESCRIPTION This function causes a new thread to be started up. For kernel-level drivers, the new thread is in the IOTask's address space; for user-level drivers, the thread is in the current task. The thread begins execution at *function*, which is passed *arg* as its argument.

IOFree()

SUMMARY Free memory allocated by **IOMalloc()**

DECLARED IN driverkit/generalFuncs.h

SYNOPSIS void **IOFree(**void **var*, int *numBytes*)*

DESCRIPTION This function frees memory allocated by **IOMalloc()**.

Note: You must use the same value for *numBytes* as you used for the call to **IOMalloc()** that allocated the memory you're now freeing.

IOFreeLow()

SUMMARY Free memory allocated by **IOMallocLow()**

DECLARED IN driverkit/i386/kernelDriver.h

SYNOPSIS void **IOFreeLow(**void **var*, int *numBytes***)**

DESCRIPTION This function frees memory allocated by **IOMallocLow()**.

Note: This function works only in kernel-level drivers.

IOGetDDMEntry()

SUMMARY Obtain an entry from the Driver Debugging Module

DECLARED IN driverkit/debugging.h

SYNOPSIS int **IOGetDDMEntry(**int *entry*, int *outStringSize*, char ***outString*, ns_time_t ***timestamp*, int ****cpuNumber***)**

DESCRIPTION Returns in *outString* an entry from the DDM. The *entry* argument should indicate which entry to return, counting backwards from the most recent entry. The *timestamp* argument is set to a value indicating the time at which the entry was logged. The *cpuNumber* argument is set to the number of the CPU that the retrieved entry is associated with.

RETURN Returns a nonzero value if the specified entry doesn't exist. Otherwise, returns zero.

IOGetDDMMask()

SUMMARY Returns the specified bitmask word

DECLARED IN driverkit/debugging.h

SYNOPSIS unsigned **IOGetDDMMask(**int *index***)**

DESCRIPTION This is typically not used by drivers; it provides a procedural means of obtaining a specified bitmask value. For performance reasons, the macros that filter and call **IOAddDDMEntry()** typically read the index words directly (the **IODDMMasks** array is a global variable).

IOGetObjectForDeviceName()

SUMMARY Obtain the **id** of a kernel device, given its name

DECLARED IN driverkit/kernelDriver.h

SYNOPSIS IOReturn **IOGetObjectForDeviceName(**IOString *deviceName*, id ***deviceId***)**

DESCRIPTION This function provides a simple mapping of device names to objects. Since this is valid only at kernel level, no security mechanism is provided; any kernel code can get the **id** of any kernel IODevice.

Note: This function works only in kernel-level drivers.

RETURN Returns IO_DR_NOT_ATTACHED if *deviceName* isn't found; otherwise returns IO_R_SUCCESS.

IOGetTimestamp()

SUMMARY Obtains a microsecond-accurate current timestamp

DECLARED IN driverkit/generalFuncs.h

SYNOPSIS void **IOGetTimestamp(**ns_time_t ***nsp*)**

DESCRIPTION This function obtains a quick, microsecond-accurate, system-wide timestamp.

IOHostPrivSelf()

SUMMARY Returns the kernel I/O task's version of the privileged host port

DECLARED IN driverkit/kernelDriver.h

SYNOPSIS port_t **IOHostPrivSelf()**

DESCRIPTION This function is necessary because the Mach function **host_priv_self()** doesn't work at kernel level.

Note: This function works only in kernel-level drivers. In user-level drivers, use **host_priv_self()** instead.

IOInitDDM()

SUMMARY Initialize the Driver Debugging Module

DECLARED IN driverkit/debugging.h

SYNOPSIS Kernel level: void **IOInitDDM(**int *numBufs*)**
User level: void **IOInitDDM(**int *numBufs*, char **serverPortName*)

DESCRIPTION This function must be called once by your driver before calling any other DDM functions.

IOInitGeneralFuncs()

SUMMARY Initialize the general-purpose functions

DECLARED IN driverkit/generalFuncs.h

SYNOPSIS void **IOInitGeneralFuncs()**

DESCRIPTION Each user-level driver must call **IOInitGeneralFuncs()** once before calling any other functions declared in the **driverkit/generalFuncs.h** header file.

Note: Kernel-level drivers don't need to call this function, because it's automatically called by the kernel.

IOIsAligned()

SUMMARY Determine whether an address is aligned

DECLARED IN driverkit/align.h

SYNOPSIS unsigned int **IOIsAligned(***address*, *bufferSize***)**

DESCRIPTION This macro returns a nonzero value if *address* is a multiple of *bufferSize*; otherwise, it returns 0.

IOLog()

SUMMARY Adds a string to the system log

DECLARED IN driverkit/generalFuncs.h

SYNOPSIS void **IOLog(**const char ***format*, ...**)**

DESCRIPTION This is the Driver Kit's substitute for **printf()**; its implementation is similar to **syslog()**. **IOLog()** logs the string to **/usr/adm/messages** by default; you can specify another destination in the configuration file **/etc/syslog.conf**. The arguments are stdargs, just as for **printf()**. This function doesn't block on single-processor systems. It runs at level LOG_ERR and its facility is kern.

SEE ALSO **printf**(3) UNIX manual page, **syslog**(3) UNIX manual page

IOMalloc()

SUMMARY Standard memory allocator

DECLARED IN driverkit/generalFuncs.h

SYNOPSIS void ***IOMalloc(**int *numBytes***)**

DESCRIPTION This function causes *numBytes* bytes of memory to be allocated; a pointer to the memory is returned. No guarantees exist as to the alignment or the physical contiguity of the allocated memory, but when **IOMalloc()** is called at kernel-level, the allocated memory is guaranteed to be wired down. Memory allocated with **IOMalloc()** should be freed with **IOFree()**.

Warning: If no memory is available, **IOMalloc()** blocks until it can obtain memory. For this reason, you shouldn't call **IOMalloc()** from a direct interrupt handler.

Drivers that can control (directly or indirectly) disks, network cards, or other devices used by a file system can run into a deadlock situation if they use **IOMalloc()** during I/O. This deadlock can occur when the pageout daemon attempts to free memory by moving pages out to disk. When the pageout daemon requests this I/O and the driver uses **IOMalloc()** to request more memory than is available,

IOMalloc() blocks. The result is deadlock: the driver can't perform the I/O until memory is freed, and the memory can't be freed by the pageout daemon until the I/O happens. In general, a driver can avoid this deadlock by not allocating large amounts of memory during I/O. For example, allocating less than 100 bytes is safe, but allocating 8K bytes is very unsafe.

IOMallocLow()

SUMMARY Allocates memory in the low 16MB of the computer's memory range

DECLARED IN driverkit/i386/kernelDriver.h

SYNOPSIS void ***IOMallocLow(**int *numBytes***)**

DESCRIPTION This function acts like **IOMalloc()**, except that the allocated range of memory is guaranteed to be in the low 16MB of system memory and to be physically contiguous. This function is provided because some cards for Intel-based computers must be mapped to low memory. Memory allocated with **IOMallocLow()** should be freed with **IOFreeLow()**.

Note: This function works only in kernel-level drivers running on Intel-based computers.

IOMapPhysicalIntoIOTask

SUMMARY Map a physical address range into your IOTask's address space

DECLARED IN driverkit/kernelDriver.h

SYNOPSIS IOReturn **IOMapPhysicalIntoIOTask**(unsigned *physicalAddress*,
unsigned *length*,
vm_address_t **virtualAddress*)

DESCRIPTION This function maps a range of physical memory into your IOTask. It returns the virtual address at which the range is mapped in the virtualAddress argument.

Note: This function works only in kernel-level drivers.

RETURN Returns an error if the specified physical range could not be mapped; otherwise, returns IO_R_SUCCESS.

SEE ALSO **IOUnmapPhysicalFromIOTask()**

IONsTimeFromDDMMsg()

SUMMARY Extracts the time from a Driver Debugging Module message

DECLARED IN driverkit/debuggingMsg.h

SYNOPSIS ns_time_t **IONsTimeFromDDMMsg(**IODDMMsg **msg***)**

DESCRIPTION This inline function combines the **timestampHighInt** and **timestampLowInt** fields from *msg* and returns the result.

IOPanic()

SUMMARY Panic or dump memory after logging a string to the console

DECLARED IN driverkit/generalFuncs.h

SYNOPSIS void **IOPanic(**const char ***reason*****)**

DESCRIPTION The *reason* argument is logged to the console, after which either a kernel panic (if in kernel space) or a memory dump (if in user space) occurs.

Note: Use of this function is an extreme measure. Use **IOPanic()** only when continued execution may cause system corruption.

IOPhysicalFromVirtual()

SUMMARY Find the physical address corresponding to a virtual address

DECLARED IN driverkit/kernelDriver.h

SYNOPSIS IOReturn **IOPhysicalFromVirtual(**vm_task_t *task*, vm_address_t *virtualAddress*, unsigned int ***physicalAddress*****)**

DESCRIPTION This function gets the physical address (if any) that corresponds to *virtualAddress*. It returns IO_R_INVALID_ARG if no physical address corresponds to *virtualAddress*. On success, it returns IO_R_SUCCESS. If *virtualAddress* is in the current task, then the *task* argument should be set to **IOVmTaskSelf()**. This function will never block. Use this function only to find the physical address of wired down memory since the physical address of unwired down memory might change over time.

Note: This function is available only at kernel level. This function shouldn't be used in a custom interrupt handlerÐit can't run at the interrupt level.

IOReadRegister(), IOWriteRegister(), IOReadModifyWriteRegister()

SUMMARY Read or write values of display registers

DECLARED IN driverkit/i386/displayRegisters.h

SYNOPSIS unsigned char **IOReadRegister(** IOEISAPortAddress *port*, unsigned char *index***)**
void **IOWriteRegister(** IOEISAPortAddress *port*, unsigned char *index*, unsigned char *value***)**
void **IOReadModifyWriteRegister(** IOEISAPortAddress *port*, unsigned char *index*, unsigned char *protect*, unsigned char *value***)**

DESCRIPTION These inline functions perform operations commonly used to read or write display registers. **IOReadRegister** reads and returns the value of the register specified by *port* and *index*. **IOWriteRegister()** writes *value* to the register specified by *port* and *index*. **IOReadModifyWriteRegister()** reads the specified register, zeroes every bit that isn't set in the *protect* mask, sets every bit that's set in *value*, and sets the register to the new value. When the *protect* mask is zero, the effect is to set the register to *value*.

Note: These functions are supported only on Intel-based computers.

IORemoveFromBdevsw(), IORemoveFromCdevsw(), IORemoveFromVfssw()

SUMMARY Remove UNIX-style entry points from a device switch table

DECLARED IN driverkit/devsw.h

SYNOPSIS void **IORemoveFromBdevsw(**int *bdevswNumber***)**
void **IORemoveFromCdevsw(**int *cdevswNumber***)**
void **IORemoveFromVfssw(**int *vfsswNumber***)**

DESCRIPTION These functions remove a device from a device switch table, replacing it with a null entry.

Note: You should use IODevice's **removeFromBdevsw** and **removeFromCdevsw** methods instead of **IORemoveFromBdevsw()** and **IORemoveFromCdevsw()**, whenever possible.

SEE ALSO **IOAddToBdevsw()**, **IOAddToCdevsw()**, **IOAddToVfssw()**

IOResumeThread()

SUMMARY Resume the execution of a thread suspended with **IOSuspendThread()**

DECLARED IN driverkit/generalFuncs.h

SYNOPSIS void **IOResumeThread(**IOThread *thread***)**

DESCRIPTION This function causes the execution of a suspended thread to continue.

IOScheduleFunc()

SUMMARY Arrange for the specified function to be called at a certain time in the future

DECLARED IN driverkit/generalFuncs.h

SYNOPSIS void **IOScheduleFunc(**IOThreadFunc *function*, void ***arg*, int *seconds***)**

DESCRIPTION This function causes *function* to be called in *seconds* seconds, with *arg* as *function*'s argument. The call to *function* occurs in the context of the caller's task, but in a thread that is unique to the Driver Kit. The call to *function* can be cancelled with **IOUnscheduleFunc()**.

Note: The kernel version of **IOScheduleFunc()** performs the callback in the kernel task's context, not the I/O Task context. One consequence is that *function* can't send Mach messages with **msg_send()**; it needs to use **msg_send_from_kernel()** instead, as described in Chapter 2.

IOSendInterrupt()

SUMMARY Arrange for an interrupt message to be sent

DECLARED IN driverkit/IODirectDevice.h

SYNOPSIS void **IOSendInterrupt(**void **identity*, void **state*, unsigned int *msgId***)**

DESCRIPTION This function is useful if you need to handle interrupts directlyÐfor example, because of a timing constraint in the hardwareÐbut don't wish to give up the advantages of interrupt notification by messages. To handle interrupts directly, you must implement the **getHandler:level:argument:forInterrupt:** message of IODirectDevice.

The *msgId* argument specifies the message ID of the interrupt message that will be sent. This should be IO_DEVICE_INTERRUPT_MSG unless the driver's documentation specifies otherwise. The *identify* and *state* arguments should be set to the values that the interrupt handler received in its own arguments. For example (italicized text delineated in angle brackets, that is << >>, is to be filled in with device-specific code):

```
static void myInterruptHandler(void *identity, void *state,
    unsigned int arg)
{
    << handle the interrupt >>
    IOSendInterrupt(identity, state, IO_DEVICE_INTERRUPT_MSG);
}
```

SEE ALSO **IODisableInterrupt(), IOEnableInterrupt()**

IOSetDDMMask()

SUMMARY Set specified bitmask word to specified value

DECLARED IN driverkit/debugging.h

SYNOPSIS void **IOSetDDMMask(**int *index*, unsigned int *bitmask*)

DESCRIPTION This is typically used by individual user-level drivers at initialization time, if then. Subsequently, it is usually used only by the Driver Debugging Module's server thread to change the current bitmask value.

The *index* argument is an index into **IODDMMasks**, which is an array of **unsigned int**. Each entry of the array contains 32 mask bits.

IOSetUNIXError()

SUMMARY Explicitly return an error value from a UNIX-style driver

DECLARED IN driverkit/kernelDriver.h

SYNOPSIS void **IOSetUNIXError(**int *errno***)**

DESCRIPTION Most UNIX-style drivers don't need to use this function. However, those that explicitly set the caller's errno can use this function to do so. This function is used when the caller executes as a result of a UNIX-style entry point.

Note: This function works only in kernel-level drivers.

IOSleep()

SUMMARY Sleep for indicated number of milliseconds

DECLARED IN driverkit/generalFuncs.h

SYNOPSIS void **IOSleep(**unsigned int *milliseconds***)**

DESCRIPTION This function causes the caller to block for the indicated number of milliseconds.

IOSuspendThread()

SUMMARY Suspend the execution of a thread started with **IOForkThread()**

DECLARED IN driverkit/generalFuncs.h

SYNOPSIS void **IOSuspendThread(**IOThread *thread***)**

DESCRIPTION This function causes the execution of a running thread to pause. The thread can be resumed with **IOResumeThread()**.

IOUnmapPhysicalFromIOTask

SUMMARY Unmap a physical address range from your IOTask's address space

DECLARED IN driverkit/kernelDriver.h

SYNOPSIS IOReturn **IOUnmapPhysicalFromIOTask**(vm_address_t *virtualAddress*, unsigned *length*)

DESCRIPTION This function unmaps a range of memory that was mapped with **IOMapPhysicalIntoIOTask()**. You should use this to destroy a mapping when you no longer need to use it.

Note: This function works only in kernel-level drivers.

RETURN Returns an error if the specified virtual range was not mapped by **IOMapPhysicalIntoIOTask()**; otherwise, returns IO_R_SUCCESS.

SEE ALSO **IOMapPhysicalIntoIOTask()**

IOUnscheduleFunc()

SUMMARY Cancel a request made with **IOScheduleFunc()**

DECLARED IN driverkit/generalFuncs.h

SYNOPSIS void **IOUnscheduleFunc(**IOThreadFunc *function*, void ***arg***)**

DESCRIPTION This function removes a request made using **IOScheduleFunc()** from the current list of pending requests. An error will be logged to the console if the specified *function*/*arg* pair is not currently registered.

IOVmTaskCurrent()

SUMMARY Returns the **vm_task_t** of the current task

DECLARED IN driverkit/kernelDriver.h

SYNOPSIS vm_task_t **IOVmTaskCurrent()**

DESCRIPTION Returns the **vm_task_t** for the current task. The only reason to use this function is to perform DMA to user space memory transfers in a UNIX-style driver.

Note: This function works only in kernel-level drivers.

SEE ALSO **IOVmTaskSelf()**

IOVmTaskForBuf()

SUMMARY Returns the **vm_task_t** associated with a **buf** structure

DECLARED IN driverkit/kernelDriver.h

SYNOPSIS vm_task_t **IOVmTaskForBuf(**struct buf **buffer***)*

DESCRIPTION Block drivers use this function to determine the task for which they're doing I/O. The value returned by this function is used in calls to **IOPhysicalFromVirtual()**, which returns an address that's used in IODirectDevice's **createDMABufferFor:...** method.

Note: This function works only in kernel-level drivers.

IOVmTaskSelf()

SUMMARY Obtain the **vm_task_t** of the kernel

DECLARED IN driverkit/kernelDriver.h

SYNOPSIS vm_task_t **IOVmTaskSelf()**

DESCRIPTION This function is used to obtain the kernel's **vm_task_t**, which is the **vm_task_t** for memory allocated with **IOMalloc()**. This function is required because the type definition of **vm_task_t** at kernel level is different from that of **vm_task_t** at user level.

Note: This function works only in kernel-level drivers.