Frename $56 Rename an existing file or folder

Class: GEMDOS *Category: File Manipulation*

SYNOPSIS

```
    #include  <osbind.h>

    error  =  Frename(zero,old,new);

    long  error;                error  status
    short  zero;                must  be  zero
    const  char  *old;          old  name
    const  char  *new;          new  name
```

DESCRIPTION

Frename renames the file old to the name new. Note that these files *do not* have to be in the same directory, but must be on the same physical device.

Under TOS 1.4 and above the Frename function may also be applied to a directory, however these may not be moved about the tree structure.

The parameter zero *must* be passed as the value 0.

RETURNS

Frename returns zero if the operation was completed successfully, or a negative error code if a problem occurred.

SEE

rename

CAVEATS

Under 1.0 and 1.2 of TOS it is not possible to rename folders, but beware of older documentation which incorrectly states that files may not be renamed up and down the directory structure.

If you attempt to rename a file you have open the file is *neither* closed *nor* is its handle released. If you continue to use this handle there may be disastrous consequences.

Fseek $42 Seek to a new file position

Class: GEMDOS *Category: File Manipulation*

SYNOPSIS

```
    #include  <osbind.h>

    apos  =  Fseek(rpos,handle,mode);

    long  apos;                 current  file  position
    long  rpos;                 new  offset
    short  handle;              file  handle  to  seek  on
    short  mode;                seek  mode
```

DESCRIPTION

The Fseek function repositions the file pointer of the file associated with handle. The seek mode is the same as for lseek as follows (defined in stdio.h):

Mode	Meaning
0 (SEEK_SET)	The rpos argument is the number of bytes from the beginning of the file. This value must be positive.
1 (SEEK_CUR)	The rpos argument is the number of bytes relative to the current position. This value can be positive or negative.
2 (SEEK_END)	The rpos argument is the number of bytes relative to the end of the file. This value must be negative or zero.

Note that for mode SEEK_CUR rpos can be positive or negative, but apos is always the actual (positive) position relative to the beginning of file.

RETURNS

If the operation is successful, the function returns the actual positive file position, which is a long integer. Otherwise a negative error code is returned.

SEE

_dseek, lseek

Fsfirst, Fsnext $4E, $4F — Find directory entry

Class: GEMDOS — *Category: File Manipulation*

SYNOPSIS

```
#include  <osbind.h>

err  =  Fsfirst(name,attr);  Find  first  directory  entry
err  =  Fsnext();            Find  next  directory  entry

long  err;                   0  if  successful
const  char  *name;          file  name  or  pattern
short  attr;                 file  attribute  bits
```

DESCRIPTION

These functions search a directory for entries that match the specified file name or file name pattern. The Fsfirst function locates the first matching file. Then successive calls to Fsnext locate additional matching files.

The name argument must be a null-terminated string specifying the drive, path, and name of the desired file. The drive and path can be omitted, in which case the current directory will be searched. You can use the GEMDOS * and ? characters for pattern matching in the name portion. For example, xy*.b will locate files in the current directory that begin with xy and have b as their extension.

The attr argument specifies which file types are to be included in the search. The following bits are used:

Bit	Meaning
0	Read-only flag
1	Hidden file flag
2	System file flag
3	Volume label flag
4	Subdirectory flag

The information found is placed into the current DTA buffer. This is equivalent to the FILEINFO structure from dos.h defined as:

```
struct  FILEINFO
{
   char  resv[21];        /*  reserved  */
   char  attr;            /*  actual  file  attribute  */
   long  time;            /*  file  time  and  date  */
   long  size;            /*  file  size  in  bytes  */
   char  name[FNSIZE];    /*  file  name  */
};
```

RETURNS

The Fsfirst function returns zero if successful, or a negative error code (e.g. if no files matching were found). Fsnext returns 0 when successful, ENMFIL (-49) when no more files are available, or some other negative error code if an error occurred.

SEE

dfind, dnext, Fgetdta

EXAMPLE

```
/*
 *  show  the  files  in  a  given  directory
 */

#include  <dos.h>
#include  <osbind.h>

void  showdir(const  char  *name)
{
   struct  FILEINFO  info;

   Fsetdta(&info);
   if  (!Fsfirst(name,0))
     do
     {
       puts(info.name);
     }  while  (!Fsnext());
}
```

Fwrite $40

Write to an open file

Class: GEMDOS

Category: File Manipulation

SYNOPSIS

```
#include  <osbind.h>

len  =  Fwrite(handle,count,buf);

long  len;                  length written to file
short  handle;              file handle
long  count;                length to write
const  void  *buf;          buffer to write from
```

DESCRIPTION

This Fwrite function writes to a file given by handle. count characters are written to the file from a buffer pointed to by buf. The process stops when either count characters have been written, or an error is encountered.

Note that this call is recommended as it is the sole output method which is consistent across all versions of TOS when used with redirection.

RETURNS

Fwrite returns the number of characters successfully written, or a negative error code if a serious error occurred. Note that if disk full occurs this is indicated by len not equal to count; an error is not explicitly returned.

SEE

Fcreat, Fclose, Fread

CAVEATS

Under 1.0 and 1.2 of TOS attempting to use Fwrite with count equal to zero will hang the system.

Malloc $48

Allocate a block of memory from the GEMDOS pool

Class: GEMDOS

Category: Memory Allocation

SYNOPSIS

```
#include  <osbind.h>

base  =  Malloc(amount);

void  *base;                base of block allocated
long  amount;               amount of memory requested
```

DESCRIPTION

The Malloc function is used to obtain blocks of memory from the GEMDOS free memory pool. The amount of memory required is passed in amount, and the base of the block allocated is returned in base. If no memory is available a NULL pointer is returned.

To determine the size of the largest free block in the system, the value -1 may be used for amount, when the pointer returned should be cast to a long value giving the size of the block. Note that it is the size of the largest free block that is returned, and *not* the total free memory in the OS pool.

RETURNS

Malloc returns the base of the memory block to use or NULL if insufficient memory was available. If amount is equal to -1 then the size of the largest block is returned.

SEE

Mfree, Mshrink, malloc

CAVEATS

Under 1.0 and 1.2 of TOS there is a limit of 20 active blocks of Malloc'ed memory per process. Exceeding this limit may cause GEMDOS to fail in a disastrous manner. Note that this limit *includes* any blocks required by other parts of the operating system, in particular virtual workstations and file selectors require GEMDOS memory and so you should consider limiting your own allocations to, say, 16 blocks.

Under TOS 1.4 and above the limit on blocks is less problematic (and the system will halt safely if the situation were to occur), however there are still limits and so you should always use an *internal* memory manager such as the C library malloc.

_mediach

Force media change on a logical device

Class: Lattice *Category: Device I/O*

SYNOPSIS

```
#include  <osbind.h>

status=_mediach(dev);

int  error;      error  status
int  dev;        device  to  force  media  change  on
```

DESCRIPTION

The _mediach function is used to force a media change on a device. It is normally used prior to calling the BIOS function Getbpb to ensure that GEMDOS cache consistency is maintained.

The parameter dev gives the number of the logical device to force the change on, 0 means drive A, 1 drive B, etc.

Note that this function should *always* be called prior to Getbpb otherwise GEMDOS data loss is almost inevitable.

RETURNS

_mediach normally returns 0 to indicate no error. It returns 1 to indicate an error situation, if this occurs you should *immediately* stop any disk I/O since GEMDOS has almost certainly suffered an internal failure.

SEE

Getbpb, Mediach

Mfree 349

Release a block of memory to the GEMDOS pool

Class: GEMDOS *Category: Memory Allocation*

SYNOPSIS

```
#include  <osbind.h>

error  =  Mfree(base);

long  error;                error  return
void  *base;                base  of  block  allocated
```

DESCRIPTION

The Mfree function is used to return blocks of memory allocated via Malloc to the GEMDOS free memory pool. The base of the block to return is passed in base.

RETURNS

Mfree returns 0 if the block was successfully freed, or a negative error code if a problem occurred (e.g. freeing a block which was not allocated).

SEE

Malloc, Mshrink, free

Mshrink $4A

Shrink size of allocated block

Class: GEMDOS

Category: Memory Allocation

SYNOPSIS

```
    #include  <osbind.h>

    error  =  Mshrink(base,size);

    long  error;            error  return
    void  *base;            base  of  block  allocated
    long  size;             new  size  of  block
```

DESCRIPTION

The Mshrink function is used to reduce the size of an allocated block of GEMDOS memory. base points to a block of allocated memory and size gives the new size that is requested for it.

Note that this function is most often used to reduce the size of a programs TPA. when first started, so that memory is available for subsequent Mallocs.

RETURNS

Mshrink returns 0 if the size of the block was successfully changed, or a negative error code if a problem occurred (e.g. attempting to enlarge a block).

SEE

Malloc, Mfree, realloc

CAVEATS

Although the interface to this function suggests it may be used to enlarge a block this does not work under all current versions of the OS, returning the error code EGSBF, 'SetBlock Failure due to Growth restrictions'.

Pexec $4B

Create/Execute process

Class: GEMDOS

Category: Process Creation

SYNOPSIS

```
    #include  <osbind.h>

    error  =  Pexec(mode,path,tail,env);

    long  error;               error  return
    short  mode;               Pexec  mode
    const  char  *path;        path  of  program  to  execute
    const  char  *tail;        command  line
    const  char  *env;         pointer  to  environment
```

DESCRIPTION

Pexec provides facilities for a program to create basepages, load programs and execute them.

path is a pointer a string giving the filename of the program to execute. If path does not specify a drive the current drive is used, similarly if no pathname is specified the current path is used. Note that any filename extension must be explicitly specified.

tail is a pointer to a length prefixed string, i.e. tail(0) contains the length of the string starting at tail(1), the total length of the string (including the length byte) may not exceed 126 bytes. Note that when copying this string GEMDOS copies 126 bytes or up to a NULL character, which ever is first.

env contains a pointer to the environment to be passed to the child process. If this pointer is NULL then the child inherits a copy of the parents environment. GEMDOS obtains a block of memory using Malloc into which it copies the child processes environment.

The mode parameter determines what function the command performs. The following mode values are allowed:

Value	Meaning
0	Create a basepage, load program into the basepage, execute program returning program's termination code when the program completes.
3	Create a basepage and load program into it. The value returned is the address of the base page created.

4	Execute program already loaded. For this mode path and env are unused (pass NULL for these). tail holds the address of the program to execute. The value returned is the program termination code. Note that the TPA and environment are *not* freed after running the program.
5	Create a basepage. For this mode path is unused (pass NULL for this), tail and env have there normal meanings. The value returned is the address of the base page created.
6	Execute program already loaded. For this mode path and env are unused, and tail holds the address of the program to execute. The value returned is the program termination code. Unlike mode 4, the TPA and environment *are* freed after executing the child process. Note the warning below about this mode.

Note that the basepage structure is described in the C library manual and also in the basepage.h header file.

RETURNS

Pexec returns values dependent on the mode argument. For all modes a *longword* negative value is an error indication, positive values are as indicated above. Note that when Pexec returns an exit code from a program it has executed the top 16 bits are zero, you may also find it useful to note that if a program is aborted via Ctrl-C then the return code is 0xffe0.

SEE

Pterm0, Pterm, Ptermres, Mshrink

CAVEATS

Pexec mode 6 is only available on GEMDOS version 0.21 (TOS 1.4) and above.

Pterm, Pterm0 $4C , $00 Terminate a process

Class: GEMDOS *Category: Process Creation*

SYNOPSIS

```
#include  <osbind.h>

Pterm(ret);
Pterm0();

short  ret;   error code to return to parent
```

DESCRIPTION

These functions immediately terminate the current process. For Pterm, a return status is passed in ret, whilst Pterm0 always gives a zero exit status to the parent (note that Pterm0 is exactly equivalent to Pterm(0)). Prior to terminating, GEMDOS makes a call through extended vector 0x102 (etv_term) so that a program may perform last minute clean up.

Any files still open which were opened by the process are closed, in addition all standard files (handles 0 to 5) are closed, note that this *includes* standard files inherited from the parent process. Any memory not released by the process is returned to the OS memory pool.

RETURNS

The function does not (normally) return.

SEE

Pexec, Ptermres, Setexc, onbreak

Ptermres $31 Terminate and stay resident (TSR)

Class: GEMDOS *Category: Process Creation*

SYNOPSIS

```
        #include  <osbind.h>

        Ptermres(keep,ret);

        long  keep;  length of process to keep
        short  ret;  error code to return to parent
```

DESCRIPTION

Ptermres is similar to Pterm, but rather than releasing the memory allocated by the process into the OS pool, it is retained by the process.

Ptermres retains keep bytes of the process (from the start of the base page) in memory. Note that this is exactly equivalent to using Mshrink on the basepage. Any additional memory which has been obtained by Malloc is also retained.

The process is then terminated as if by Pterm(ret).

Programs which terminate using this method are usually known as TSRs and are usually used to patch the operating system in some manner or other.

RETURNS

The function does not (normally) return.

SEE

Pexec, Pterm, Setexc, onbreak

CAVEATS

Because Ptermres implicitly calls Pterm, any open files are closed and so lost to the process.

This call actually removes the processes memory from the allocation table of GEMDOS, but does not place it into the free table, thus any memory so retained is *permanently* lost, i.e. a subsequent Pterm or Mfree call will not return it to GEMDOS.

Super $20 Get/Set/Inquire supervisor mode

Class: GEMDOS *Category: System Manipulation*

SYNOPSIS

```
        #include  <osbind.h>

        oldssp  =  Super(stack);

        void  *oldssp;          old system stack pointer
        void  *stack;           system stack request value
```

DESCRIPTION

The Super function allows you to alter the state of the processor. If stack is NULL then the processor is placed into supervisor mode and the old supervisor stack returned in oldssp. Note that the supervisor stack is then pointed at the user stack.

Otherwise if stack is non-NULL, this is taken to be an old supervisor stack value which is reloaded into the supervisor stack pointer and the processor placed back into user mode.

To allow interrogation of the processor state, the special value of stack==1, causes the value returned in oldssp to be 0 if the processor is in user mode, or -1 if in supervisor mode. Beware of some older documentation which states that stack should be -1 to interrogate the processor mode. Using this value will result in a system crash.

RETURNS

As noted above.

SEE

Supexec

CAVEATS

Whilst in supervisor mode the AES *may not* be called. It *always* assumes that it has been called from user mode and saves registers on the user stack.

Also beware that entry to supervisor mode and exit from it *must* occur in the same function. You may not call a routine to enter supervisor mode and then call a second routine to leave it. Failure to enter and leave supervisor mode within the same stack frame will cause the stack pointer to become randomly corrupted.

Sversion $30 Get GEMDOS version number

Class: GEMDOS *Category: System Manipulation*

SYNOPSIS

```
#include <osbind.h>

version = Sversion();

unsigned short version;   GEMDOS version number
```

DESCRIPTION

Sversion returns the version number of GEMDOS. Note that this is *not* the same as the TOS or AES version numbers. The value returned in version is byte swapped, so that the low byte gives the major version number, whilst the high byte gives the minor version number. The currently used values are:

Major	Minor	Name
0	19	ROM TOS (1.0), Blitter TOS (1.2)
0	21	Rainbow TOS (1.4), STE TOS (1.6)

RETURNS

As noted above.

SEE

_tos, appl_init

EXAMPLE

```
/*
 * print the GEMDOS version number
 */

#include <osbind.h>
#include <stdio.h>

int main(void)
{
  unsigned short ver=Sversion();

  printf("GEMDOS version=%d.%d\n",ver&0xff,ver>>8);
  return 0;
}
```

Tgetdate, Tsetdate $2A-$2B Get/Set GEMDOS date

Class: GEMDOS *Category: Date and Time*

SYNOPSIS

```
#include <osbind.h>

date = Tgetdate();
error = Tsetdate(date);

long error;                error status
unsigned short date;       packed date
```

DESCRIPTION

Tgetdate returns the current date in GEMDOS format. This is packed as follows:

Bits	Contents
0-4	Day (0 to 31)
5-8	Month (1 to 12)
9-15	Year-1980 (0 to 127)

The associated function Tsetdate sets the current date to the packed date which is its parameter.

RETURNS

Tgetdate returns the current packed time, whilst Tsetdate returns 0 for valid dates or an error code for *obviously* invalid dates.

SEE

Tgettime, Tsettime, Gettime, Settime, ftunpk, ftpack, time

CAVEATS

Under TOS 1.0 Tsetdate does not inform the BIOS of the date change, hence it does not change the IKBD clock or any battery-backed clock.

Tgettime, Tsettime $2C ,$2D Get/Set GEMDOS time

Class: GEMDOS *Category: Date and Time*

SYNOPSIS

```
#include <osbind.h>

time  = Tgettime();
error  = Tsettime(time);

long error;                 error status
unsigned short time;        packed time
```

DESCRIPTION

Tgettime returns the current time in GEMDOS format. This is packed as follows:

Bits	Contents
00-04	Second/2 (0 to 29)
05-10	Minute (0 to 59)
11-15	Hour (0 to 23)

The associated function Tsettime sets the current time to the packed time which is its parameter.

RETURNS

Tgettime returns the current packed time, whilst Tsettime returns 0 for valid times or an error code for *obviously* invalid times.

SEE

Tgetdate, Tsetdate, Gettime, Settime, ftunpk, ftpack, time

CAVEATS

Under TOS 1.0 Tsettime does not inform the BIOS of the time change, hence it does not change the IKBD clock or any battery-backed clock.

5 BIOS Library

This section describes the BIOS library supplied with the Lattice C compiler. To access the facilities of the BIOS you should #include the file osbind.h into your program.

The BIOS provides the low level console and disk manipulation functions for GEMDOS. In general you should have no need to call this level of the OS as it provides facilities which are not always compatible with GEMDOS. Note that the exception to this is when using the serial port, for which the BIOS should always be used due to problems in GEMDOS.

Like GEMDOS the BIOS uses a consistent set of prefixes for its naming, these are:

Prefix	Function
Bcon	Direct access to character device input/output.
Drv	Disk management.
Get	System parameter block inquiry.
Kb	Low level keyboard driver information.
Med	Media inquiry functions.
L, R	Device logical sector access.
S	System inquiry and manipulation.
T	Time and date functions.

All functions in the BIOS library are available either through the original Atari macro based definitions or through the inline code capability of the Lattice C compiler. Using this facility will greatly reduce the overheads compared with the old 'stub' based method.

Bconin

Read a character from a device

Class: BIOS

Category: Console and Port I/O

SYNOPSIS

```
#include  <osbind.h>

x=Bconin(dev);

long  x;            character  obtained
short  dev;         device  to  get  character  from
```

DESCRIPTION

The Bconin function reads (without echoing) a character from the specified device. The legal values are:

Value	Meaning
0	Parallel printer port
1	Auxiliary device (the RS232 port)
2	Console device
3	MIDI port

For the console (device 2) Bconin returns the scancode in the low byte of the upper word, and the ASCII character in the low byte of the low word. This gives the format:

bits 31-24	bits 23-16	bits 15-8	bits 7-0
Shift key status	Keyboard scan code	0	ASCII value of character

Note that the shift key status is only returned if bit 3 in the system variable conterm (the character at 0x484) is set. This defaults to off.

The non-ASCII keys (e.g. the function and cursor keys) return 0 for the ASCII value, so that the scan code is used to decipher them. The shift key status gives the state of the keyboard modifiers (Shift, Ctrl, Alt etc.) and are as described under the BIOS function Kbshift.

RETURNS

Returnerar när det finns en bokstav.

As noted above.

SEE

Bconstat, Cconin, Cauxin

CAVEATS

The conterm variable is a system global so either all processes or no processes get the shift key state.

EXAMPLE

```
/*
 * display key-presses as they occur
 */

#include  <osbind.h>

int  oconterm;

int  conset(void)
{
  oconterm=*(char  *)0x484;
  *(char  *)0x484|=1<<3;
}

int  conunset(void)
{
  *(char  *)0x484=oconterm;
}

int  main(void)
{
  const  char  *unshift;

  unshift=*Keytbl(-1,-1,-1);
  Supexec(conset);     /* set the shift key bit */
  for  (;;)
  {
    long  x;

    x=Bconin(2);  /* get key code */
    /*  shift-shift-ctrl-alt  ends  */
    if  ((x&0x0f000000)==0x0f000000)
      break;
    printf("ASCII  code=%ld;Scan  code=%ld;Shift=%ld\n",
      x&0xff,  (x>>16)&0xff,  (x>>24)&0xff);
    /*  look  up  key  legend  in  keyboard  table  */
    printf("Key  legend='%c'\n",unshift[(x>>16)&0xff]);
  }
  Supexec(conunset);  /* reset shift key bit */
  return  0;
}
```

Bconout

Write a character to a device

Class: BIOS

Category: Console and Port I/O

SYNOPSIS

```
#include <osbind.h>

error=Bconout(dev,c);

long error;      error status
short dev;       device to send character to
short c;         character to send to device
```

DESCRIPTION

The Bconout function writes the character c to the specified device. The legal device values (dev) are:

Value	Meaning
0	Parallel printer port
1	Auxiliary device (the RS232 port)
2	Console device
3	MIDI port
4	Keyboard port (IKBD)
5	Raw screen device

RETURNS

For output to the printer, RS232, MIDI and IKBD devices, the function returns 0 to indicate failure or non-zero on success.

SEE

Bcostat, Cconout, Cauxout, Cprnout

Bconstat

Return device input status

Class: BIOS

Category: Console and Port I/O

SYNOPSIS

```
#include <osbind.h>

status=Bconstat(dev);

long status;     input status
short dev;       device to interrogate
```

DESCRIPTION

Bconstat obtains the input status of a character device. The parameter dev gives the device for which you want to know the status. The legal values are:

Value	Meaning
0	Parallel printer port
1	Auxiliary device (the RS232 port)
2	Console device
3	MIDI port

RETURNS

The value returned in status is 0 if no characters are available, or -1 if at least one character is available.

SEE

Bconin, Cconis, Cauxis

Bcostat

Check character device output status

Class: BIOS *Category: Console and Port I/O*

SYNOPSIS

```
#include <osbind.h>

status=Bcostat(dev);

long status;    output status
short dev;      device to check status of
```

DESCRIPTION

The Bcostat function checks the output status of the specified device. The legal device values (dev) are:

Value	Meaning
0	Parallel printer port
1	Auxiliary device (the RS232 port)
2	Console device
3	MIDI port
4	Keyboard port (IKBD)
5	Raw screen device

RETURNS

The function returns 0 to indicate that the device is not ready to receive, or non-zero to indicate that a character may be sent without waiting.

SEE

Bconout, Cconos, Cauxos, Cprnos

Drvmap

Return bitmap of mounted drives

Class: BIOS *Category: Device I/O*

SYNOPSIS

```
#include <osbind.h>

bmap=Drvmap();

unsigned long bmap;     bitmap of mounted drives
```

DESCRIPTION

The Drvmap function returns a bit map of drives mounted (i.e. available) on the system. Each bit represents a single drive which exists if set. Bit 0 corresponds to drive A, bit 1 to drive B etc.

Note that on a system with only a single floppy both bits 0 *and* 1 will be set, and 'virtual-disking' will be used to provide both devices.

RETURNS

The bitmap of mounted drives. Note that it is up to device drivers to update the system global _drvbits if they are to be recognised by the system.

SEE

Dsetdrv

EXAMPLE

```
/*
 * List the mounted drives
 */

#include <osbind.h>
#include <stdio.h>

int main(void)
{
  unsigned long bmap;
  int i;

  bmap=Drvmap();                /* fetch the bitmap */
  for (i=0; i<32; i++)          /* scan over the bits */
    if (bmap&1<<i)              /* check a bit */
      printf("Drive %c: is mounted\n",i+'A');
  return 0;
}
```

Getbpb

Get BIOS parameter block for a device

Class: BIOS

Category: Device I/O

SYNOPSIS

```
#include <osbind.h>

bpb=Getbpb(dev);

volatile void *bpb;          pointer to device BPB
short dev;                   device to obtain BPB for
```

DESCRIPTION

Getbpb returns a pointer to the BIOS parameter block for the requested device dev. bpb points to structure of the form:

```
typedef struct
{
  short  recsiz;   bytes per sector
  short  clsiz;    sectors per cluster
  short  clsizb;   bytes per cluster
  short  rdlen;    length in sectors of root directory
  short  fsiz;     sectors per FAT
  short  fatrec;   record number of start of second FAT
  short  datrec;   record number of start of data
  short  numcl;    clusters per disk
  short  bflags;   bit 0==1 - 16 bit FAT, else 12 bit
} BPB;
```

Note that calling this function causes the driver to update the media-changed flag to 'not changed' for the device. If the device has changed and GEMDOS has not noticed then data may be damaged on the device. The function _mediach should be used to force GEMDOS to recognise a media change prior to calling this function.

RETURNS

The function returns a pointer to the BIOS parameter block for the device requested or NULL if the BPB could not be obtained (e.g. trying to get the BPB of an unknown device).

SEE

_mediach

CAVEATS

If a media change is not forced via _mediach prior to calling this function, data loss is almost certain to occur as GEMDOS's data caches may become invalid.

Getmpb

Size machine memory

Class: BIOS

Category: Memory Allocation

SYNOPSIS

```
#include <osbind.h>

Getmpb(mpb);

void *mpb;      pointer to prototype mpb
```

DESCRIPTION

Getmpb is used during the GEMDOS startup sequence to size the GEMDOS free memory. mpb points to a memory parameter block structure which is filled in by the call. An MPB has the form:

```
typedef struct md
{
  struct md *m_link;    next MD
  void *m_start;        start of block
  long m_length;        bytes in block
  BASEPAGE *m_own;      owner's basepage
} MD;

typedef struct mpb
{
  MD *mp_mfl;           free list
  MD *mp_mal;           allocated list
  MD *mp_rover;         roving ptr
} MPB;
```

Note that this function is called very early on in the GEMDOS startup sequence and is not useful subsequently, there are no occasions when its use is legal or desirable by a users program.

SEE

Malloc

Kbshift

Find state of keyboard 'shift' keys

Class: BIOS

Category: Console and Port I/O

SYNOPSIS

```
#include <osbind.h>

state=Kbshift(mode);

long state;                  old keyboard state
short mode;                  new state for keyboard
```

DESCRIPTION

The Kbshift function returns allows the user to read or change the state of the keyboard 'shift' keys. The parameter dev gives the new state into which the keys are to be placed. The bits and their meanings are:

Bit	Meaning (when set)
0	Right shift key down
1	Left shift key down
2	Ctrl key down
3	Alt key down
4	Caps-lock engaged
5	Clr/Home key down
6	Insert key down

If dev is set to -1 then the keyboard state is not changed and the the current state is returned.

Note that bits 5 and 6 are not the left and right mouse buttons as inferred by some documentation; they are, however, the keyboard equivalents.

RETURNS

Kbshift returns the old state of the keyboard shift bits.

EXAMPLE

```
/*
 * Force Caps-Lock on
 */

#include <osbind.h>
#include <stdio.h>

int main(void)
{
  long state;
  char buf[80];

  state=Kbshift(1<<4);     /* caps on, save old state */
  while (!feof(stdin))     /* wait for a ctrl-Z */
    gets(buf);             /* type something to test */
  Kbshift(state);          /* restore old state */
  return 0;
}
```

Mediach

Return media change status

Class: BIOS

Category: Device I/O

SYNOPSIS

```
    #include  <osbind.h>

    status=Mediach(dev);

    long  error;     changed  status
    short  dev;      device  to  obtain  status  of
```

DESCRIPTION

The Mediach function returns the 'media-change' status of the device specified by dev. This function is used by GEMDOS to detect media changes on removable media (e.g. floppy disks).

Note that if the BIOS detects a definite media-change, before GEMDOS has cleared it (via Getbpb), then it will issue a media changed error (E_CHNG).

RETURNS

The function returns a value of 0, 1 or 2 in status representing the situations:

Value	Meaning
0	Media definitely has *not* changed
1	Media *might* have changed
2	Media definitely *has* changed

SEE

Getbpb, _mediach

Rwabs, Lrwabs

Read/Write logical sectors on a device

Class: BIOS

Category: Device I/O

SYNOPSIS

```
    #include  <osbind.h>

    error=Rwabs(mode,buf,count,recno,dev);
    error=Lrwabs(mode,buf,count,dev,lrec);

    long  error;      error  status
    short  mode;      r/w  mode  to  use
    void  *buf;       pointer  to  buffer
    short  count;     number  of  sectors  to  transfer
    short  recno;     logical  sector  to  start  at
    short  dev;       device  to  use
    long  lrec;       long  logical  sector  to  start  at
```

DESCRIPTION

The Rwabs and Lrwabs functions are used to read and write sectors to and from a 'block' device. The mode parameter has bits which specify the way the operation will occur. Note that all devices do not support all bits. The bits currently used are:

Bit	Meaning
0	Write/ $\overline{\text{Read}}$ i.e. write when bit is set.
1	If set then do not affect the media change status, or check it.
2	Disable retry when set.
3	If set do not translate logical sectors to physical sectors (i.e. recno gives a physical rather than a logical sector number).

The operation is performed into a buffer pointed to by buf, which must be large enough for the operation. In logical mode it must be at least count * the logical sector size, whilst in physical mode it must be count * 512. Note that buf need not be word aligned but for reasons of efficiency it should in general be aligned in that way.

The count parameter specifies how many sectors will be transferred, and dev specifies which device the transfer is to occur on.

recno gives the first sector (logical or physical) to read/write from. If this parameter is larger than 32767 then the long Rwabs form Lrwabs should be used, where lrec has the same meaning as recno.

RETURNS

The functions return 0 on success or a negative error code on failure. Note that as a result of processing this function the critical error handler (etv_critic) may be called.

SEE

Floprd, Flopwr

CAVEATS

Bits 2 and 3 in the mode parameter are rarely supported. Also the long Rwabs form, Lrwabs, was only introduced with Atari's AHDI 3.0.

Setexc

Set exception vector

Class: BIOS *Category: Vector Handling*

SYNOPSIS

```
    #include  <osbind.h>

    old=Setexc(num,vec);

    void  (*old);()      old  vector  entry
    short  num;          vector  number  to  change
    void  (*vec)();      new  exception  handler
```

DESCRIPTION

The Setexc function is used to modify a system exception vector. num gives the number of the vector to modify. The following values are currently allowed:

Value	Vector
0-0xff	Standard 68000 exception vectors.
0x100	System timer vector (etv_timer).
0x101	Critical error handler (etv_critic).
0x102	Process terminate handler (etv_term).
0x103-0x107	Reserved.

The new vector is given in vec. If it has the value (void *)-1 then the current vector is not changed and the value simply returned.

RETURNS

The functions returns the old value of the exception handler. Note that you *must* remove all exception handlers prior to your process terminating.

Tickcal

Get system timer 'tick' interval

Class: BIOS

Category: Date and Time

SYNOPSIS

```
    #include  <osbind.h>

    tick=Tickcal();

    long  tick;     system  tick  interval
```

DESCRIPTION

Tickcal returns the system timer calibration value in milliseconds. This is the value passed to etv_timer as a parameter. For current systems it has the value 50.

RETURNS

As noted above.

6 XBIOS Library

This section describes the XBIOS library supplied with the Lattice C compiler. To access the facilities of the XBIOS you should #include the file osbind.h into your program.

The XBIOS provides the very lowest level of access in the operating system to the hardware. In general there are very few occasions when calling it is justified from a user program, and to do so, usefully, low level documentation on the hardware is required.

Unlike other parts of the OS the XBIOS has little naming consistency in its functions.

All functions in the XBIOS library are available either through the original Atari macro based definitions or through the inline code capability of the Lattice C compiler. Using this facility will greatly reduce the overheads compared with the old 'stub' based method.

Bioskeys

Reset keyboard translation tables

Class: XBIOS

Category: Keyboard Configuration

SYNOPSIS

```
#include <osbind.h>

Bioskeys();
```

DESCRIPTION

Bioskeys is used to restore the default power-up setting of the keyboard translation tables. This will normally only be required if they have been changed via Keytbl.

SEE

Keytbl

Blitmode

Get/Set blitter configuration

Class: XBIOS

Category: Graphics Configuration

SYNOPSIS

```
#include <osbind.h>

old=Blitmode(mode);

short old;       old blitter configuration
short mode;      new blitter mode
```

DESCRIPTION

Blitmode is used to detect the presence and alter the configuration of a hardware blitter. Currently only a single bit in mode is allocated, with bit 0 being set to enable the hardware blitter, or 0 to disable. Alternatively the value -1 may be used to obtain the current blitter status.

The old configuration is returned in old and has two bits of use:

Bit	Meaning when set
0	Perform blits in hardware
1	Hardware blitter is available

RETURNS

As noted above.

EXAMPLE

```
/*
 * detect the presence of a blitter and enable it
 */
#include <osbind.h>
#include <stdio.h>

int main(void)
{
  short old=Blitmode(-1);

  if (old&2)
  {
    Blitmode(old|1);
    printf("Blitter enabled\n");
  }
  else
    printf("Sorry no blitter\n");
  return 0;
}
```

Cursconf

Configure VT52 cursor

Class: XBIOS

Category: Graphics Configuration

SYNOPSIS

```
#include <osbind.h>

old=Cursconf(function,rate);

short old;          old cursor flash rate
short function;     cursor parameter to change
short rate;         new flash rate
```

DESCRIPTION

Cursconf is used to configure the VT52 cursor. function should have a value giving the parameter you wish to change:

Value	Meaning
0	Hide cursor.
1	Show cursor.
2	Enable blinking.
3	Disable blinking.
4	Set blink rate to rate.
5	Return current blink rate.

The blink rate (for mode 4 and 5) is specified in half-frame rates, i.e. 70Hz for mono, 50/60Hz for colour.

RETURNS

For modes 0-4 the return value has no meaning. In mode 5 the current cursor blink rate is returned.

CAVEATS

There is no way of obtaining the current blink or hide status of the cursor.

Dosound

Initialise sound Dæmon

Class: XBIOS

Category: Sound Functions

SYNOPSIS

```
#include <osbind.h>

Dosound(cmd);

const char *cmd; pointer to command stream
```

DESCRIPTION

Dosound is used to start a new sound sequence through the sound dæmon. cmd should point to a byte stream consisting of commands for the dæmon consisting (in general) of one byte opcode and one byte operand pairs.

Commands 0-15 select a register, the following byte is then loaded into that register.

Command 0x80 stores the next byte into a temporary register for use by command 0x81.

Command 0x81 takes three parameters. The first is a register to load with the value in the temporary register, the second a signed value to add to the temporary register and the third the final value of the temporary register. The value of the temporary register is then stored into the register mentioned and modified by the increment until the termination condition is reached.

The final command is 0x82 (in fact any value ≥0x82) which has an argument which specifies the number of ticks (50Hz) until the next command should be executed, or the special value 0 to terminate processing.

SEE

Giaccess

CAVEATS

This is an interrupt driven routine so you should not use an automatic array to hold the dæmon commands.

Flopfmt

Format a track on a floppy disk

Class: XBIOS

Category: Floppy Disk I/O

SYNOPSIS

```
#include <osbind.h>

err=Flopfmt(buf,skew,dev,spt,track,side,
                             intlv,magic,virgin);

short err;              error status
void *buf;              pointer to word aligned buffer
short *skew;            pointer to skew table
short dev;              device to read from
short spt;              sector to read
short track;            track to read from
short side;             side to read from
short intlv;            sector interleave factor
long magic;             0x87654321
short virgin;           uninitialised sector value
```

DESCRIPTION

Flopfmt is used to format a track on a floppy disk. buf is used to build up an exact image of the track and should point to a buffer of 8Kbytes. The track formatted is track on drive dev, with spt sectors per track on side side.

magic must be the value 0x87654321; this is used to ensure that formats are less likely to occur by accident. virgin is a value which is placed in the new sectors. Typically this value is 0xe5e5; note that it may not be a value which has the high nybble of either byte set (e.g. 0xf0f0 is illegal) as these would be interpreted as commands to the FDC.

The intlv parameter gives the interleave which is to be used when creating the sectors, typically this will be 1 giving consecutively sectors. If it has the special value -1 then the parameter skew is used and should point to an array of spt shorts giving the required layout of sectors (e.g. 1,6,2,7,3,8,4,9,5 for spt==9).

Flopfmt returns in buf a word list of sectors which failed during the verify phase. Note that these are not necessarily in numerical order and are 0 terminated. If no sectors failed then *(short *)buf==0;

Calling this function causes the device to enter a 'media definitely changed' state which will be indicated at the next Rwabs or Mediach call.

RETURNS

Flopfmt returns 0 if the track was successfully formatted, or a negative error code if an error occurred.

SEE

Floprd, Flopwr, Flopver, Floprate, Rwabs

CAVEATS

The skew parameter is only supported on TOS 1.2 and above. It is ignored on TOS 1.0.

EXAMPLE

```
/*
 * Format a single-sided floppy with n-sector skewing
 */

#include <osbind.h>
#include <stdio.h>
#include <string.h>

int main(void)
{
  static char buf[8192];
  int trk;
  short  skew[]={2,3,4,5,6,7,8,9,1,2,3,4,5,6,7,8,9};
  int n=2;

  for (trk=0; trk<80; trk++)
  {
    printf("\rFormatting track %02d",trk);
    if (Flopfmt(buf,&skew[8-(trk*n%9)],0,9,trk,0,
                             -1,0x87654321,0xe5e5))
      printf("\nError on track %02d\n",trk);
  }

  /* zero the buffer */
  memset(buf,0,9*512);

  /* initialise FAT and directory */
  Flopwr(buf,0L,0,1,0,0,9);
  Flopwr(buf,0L,0,1,1,0,9);

  /* build a boot sector */
  Protobt(buf,0x01000000L,2,0);

  /* and write it out */
  Flopwr(buf,0L,0,1,0,0,1);
}
```

Floprate

Set floppy disk step rate

Class: XBIOS *Category: Floppy Disk I/O*

SYNOPSIS

```
#include <osbind.h>

old=Floprate(dev,rate);

short old;        old step rate
short dev;        device to change rate for
short rate;       new step rate
```

DESCRIPTION

Floprate is used to change the track-to-track stepping rate of the floppy disk controller for each drive. The device to change the rate of is passed in dev, and the new rate in rate. rate has the values:

Value	Seek rate
0	6ms
1	12ms
2	2ms
3	3ms

Note that to simply inquire the seek rate the value -1 may be used for rate.

RETURNS

The old seek rate for the specified drive is returned in old.

CAVEATS

This function is only available on TOS 1.4 and above, for earlier versions the system variable seekrate should be used instead, but, unlike Floprate, does not allow different seek rates on each of the drives.

Floprd

Read sectors from a floppy disk

Class: XBIOS *Category: Floppy Disk I/O*

SYNOPSIS

```
#include <osbind.h>

err=Floprd(buf,junk,dev,sect,track,side,cnt);

short err;        error status
void *buf;        pointer to word aligned buffer
long junk;        unused longword
short dev;        device to read from
short sect;       first sector to read
short track;      track to read from
short side;       side to read from
short cnt;        number of sectors to read
```

DESCRIPTION

Floprd is used to read one or more sectors from a floppy disk. cnt sectors are read from device dev (0 or 1 indicating drive A or B), starting at sector sect on track track, side side into a buffer at buf. junk is not currently used and should have the value 0L for future compatibility.

Note that this function will only read consecutive physical sectors within a track and the Rwabs function should be used to obtain logical sectors.

RETURNS

Floprd returns 0 if the required number of sectors were successfully read, or a negative error code if an error occurred.

SEE

Flopwr, Flopfmt, Flopver, Floprate, Rwabs

Flopver

Verify sectors on a floppy disk

Class: XBIOS

Category: Floppy Disk I/O

SYNOPSIS

```
#include <osbind.h>

err=Flopver(buf,junk,dev,sect,track,side,cnt);

short  err;       error status
void  *buf;       pointer to 1K word aligned buffer
long   junk;      unused longword
short  dev;       device to verify on
short  sect;      first sector to verify
short  track;     track to verify
short  side;      side to verify
short  cnt;       number of sectors to verify
```

DESCRIPTION

Flopver is used to verify one or more sectors on a floppy disk. cnt sectors are verified on device dev (0 or 1 indicating drive A or B), starting at sector sect on track track, side side using the 1K buffer buf. junk is not currently used and should have the value 0L for future compatibility.

Flopver returns in buf a word list of sectors which failed. Note that these are not necessarily in numerical order and are 0 terminated. If no sectors failed then *(short *)buf==0;

RETURNS

Flopver returns 0 if all sectors were verified successfully, or a negative error code if an error occurred.

SEE

Flopwr, Flopfmt, Floprd, Floprate, Rwabs

Flopwr

Write sectors to a floppy disk

Class: XBIOS

Category: Floppy Disk I/O

SYNOPSIS

```
#include <osbind.h>

err=Flopwr(buf,junk,dev,sect,track,side,cnt);

short  err;       error status
void  *buf;       pointer to word aligned buffer
long   junk;      unused longword
short  dev;       device to write to
short  sect;      first sector to write
short  track;     track to write to
short  side;      side to write to
short  cnt;       number of sectors to write
```

DESCRIPTION

Flopwr is used to write one or more sectors to a floppy disk. cnt sectors are written to device dev (0 or 1 indicating drive A or B), starting at sector sect on track track, side side from a buffer at buf.

Note that this function will only write consecutive physical sectors and the function Rwabs should be used to write logical sectors.

If this function is used to write to track 0, sector 1 then the device will enter a 'media might have changed' state which will be indicated at the next Rwabs or Mediach call.

RETURNS

Flopwr returns 0 if the requested sectors were successfully written, or a negative error code if an error occurred.

SEE

Floprd, Flopfmt, Flopver, Floprate, Rwabs

Getrez

Find current screen mode

Class: XBIOS *Category: Graphics Configuration*

SYNOPSIS

```
#include  <osbind.h>

res=Getrez();

short  res;        current  screen  mode
```

DESCRIPTION

Getrez returns a coded value for the current screen mode. The values *currently* returned in res are:

Value	Screen mode
0	Low resolution (320x200x4)
1	Medium resolution (640x200x2)
2	High resolution (640x400x1)

RETURNS

As noted above.

SEE

v_opnwk, Setscreen

CAVEATS

You should *not* use this function except as indicated under v_opnvwk. If you do rely on this function your application will, in general, not work on large screen monitors or on the extended screen modes of the Atari TT.

If your application needs to know the size of the screen, the number of bitplanes, or other mode specific information it should interrogate the AES, VDI or Line-A for the information rather than relying on hard-coded constants based on the result of this call.

Gettime, Settime

Get/Set IKBD time

Class: XBIOS *Category: Date and Time*

SYNOPSIS

```
#include  <osbind.h>

time=Gettime();
Settime(time);

long  time;            IKBD  time  value
```

DESCRIPTION

Gettime and Settime are used to manipulate the setting of the IKBD clock. The time is packed in the same way as GEMDOS viz:

Bits	Contents
0-4	Second/2 (0 to 29)
5-10	Minute (0 to 59)
11-15	Hour (0 to 23)
16-20	Day (0 to 31)
21-24	Month (1 to 12)
25-31	Year-1980 (0 to 127)

For Settime the single parameter gives the packed time to which the IKBD clock is to be set.

RETURNS

Gettime returns the packed IKBD time.