getdfree — *Get free disk space*

SYNOPSIS

```
#include <ext.h>

getdfree(drive,info);

int drive;                  drive code, 0 => current
struct dfree*info;          disk information
```

DESCRIPTION-

This function obtains information about the specified disk drive, including the amount of free space available. If a 0 is passed as the drive number, information is obtained about the current drive. The `dfree` structure is defined in ext.h as follows:

```
struct dfree {
   unsigned long df_avail;     number of free clusters
   unsigned long df_total;     clusters per drive
   unsigned long df_bsec;      bytes per sector
   unsigned long df_sclus;     sectors per cluster
};
```

RETURNS

None.

getdisk, setdisk — *Get or set current disk drive*

SYNOPSIS

```
#include <ext.h>

drive = getdisk();
bmap = setdisk(drive);

int drive;                  drive code
int bmap;                   bitmap of mounted drives
```

DESCRIPTION

The `setdisk` function changes the current drive code. Drive code 0 corresponds to drive A, code 1 is drive B and so on.

The `getdisk` function gets the current drive code, using the same codes as `setdisk`.

RETURNS

The function `setdisk` returns a bitmap of mounted drives, bit 0 corresponds to drive A, bit 1 is drive B and so on.

The function `getdisk` returns the code of the currently selected drive.

SEE

`chgdsk, Dsetdrv, Dgetdrv, getcd, getdsk`

getftime, setftime — *Get/set file time/date*

SYNOPSIS

```
#include <ext.h>

err = getftime(fh,curtime);     get file time
err = setftime(fh,newtime);     set file time

int err;                    0 if successful
int fh;                     handle of file
struct ftime *curtime;      pointer to current time
struct ftime *newtime;      pointer to time to be set
```

DESCRIPTION

These functions manipulate the time and date stamp of the file referenced by `fh`. `getftime` fills in the structure of type `ftime` passed to it with the current file time. `settime` is passed a similar structure containing the time which you wish to set. The structure `ftime` has the type:

```
struct ftime {
   unsigned ft_hour:5;          hour
   unsigned ft_min:6;           minute
   unsigned ft_tsec:5;          seconds
   unsigned ft_year:7;          year - 1980
   unsigned ft_month:4;         month
   unsigned ft_day:5;           day
};
```

RETURNS

If the operation is successful, a value of 0 is returned. Otherwise, the return value is -1, and further error information can be found in `errno` and `_OSERR`.

SEE

`chgft, Fdatime, getft`

gettime, settime — Get/set system time

SYNOPSIS

```
#include <ext.h>

gettime(curtime);          get current system time
settime(newtime);          set current system time

struct time *curtime;      pointer to current time
struct time *newtime;      pointer to time to be set
```

DESCRIPTION

These functions manipulate the system time. `gettime` fills in the structure of type `time` passed to it with the current system time. `settime` is passed a similar structure containing the time which you wish to set. The structure `time` has the type:

```
struct time {
   unsigned char ti_min;      minute
   unsigned char ti_hour;     hour
   unsigned char ti_hund;     hundredths - always 0
   unsigned char ti_sec;      seconds
};
```

RETURNS

None.

SEE

`getdate, setdate, stime, Tgetdate, Tsetdate`

ftw.h

ftw.h contains only one function, `ftw()`. This function enumerates the members of a directory calling a user defined function for each member.

ftw — Walk a file tree

SYNOPSIS

```
#include <ftw.h>

err = ftw(root,fn,maxdir);

int err;                        error status
const char *root;               root of directory tree
int (*fn)(path,stat,type);      function called
int maxdir;                     maximum directory depth
const char *path;               path of current file
struct stat *stat;              pointer to stat block
int type;                       type of file encountered
```

DESCRIPTION

`ftw` is used to recursively traverse a directory structure. The search starts at `path` with the function `fn` called for every file/directory encountered. `maxdir` gives the maximum number of `opendir` calls which will be made, if the depth of the tree exceeds this value then the performance of the routine will be considerably reduced.

The string passed to `fn`, `path`, gives the path of the file currently being considered, whilst `stat` contains a pointer to a `struct stat` giving information about the file (see `stat`). The `type` parameter gives further information on the file:

Symbol	Meaning
FTW_F	normal file
FTW_D	directory
FTW_DNR	unreadable directory
FTW_NS	unreadable file info

Note that if the value is `FTW_NS` then the `stat` struct will not contain useful information.

The value returned by `fn` decides whether the tree walk is to continue or be terminated; the function should return zero to continue the traversal, or non-zero to terminate.

Note that `ftw` visits a directory before visiting any of its descendants.

RETURNS

The function returns the value from the last call to the user function `fn`, i.e. zero on success.

SEE

`opendir, stat`

EXAMPLE

```
#include <ftw.h>

int fn(const char *path, struct *x, int y)
{
   puts(path);
   return 0;        /* to continue the traversal */
}

int main(void)
{
   return ftw("",fn,20);
}
```

ieeefp.h

The ieeefp.h header file includes definitions for manipulating the math co-processors of the Mega ST^{E} and TT. These functions and variables allow the precision and rounding modes of the chips to be set and examined.

_FPCfpcr Floating point co-processor configuration

SYNOPSIS

```
#include <ieeefp.h>

long _FPCfpcr;          co-processor configuration
```

DESCRIPTION

The `_FPCfpcr` variable is used at program startup time to initialise the rounding and precision of the maths co-processor. The value is formed by ORing the required flags together. The flags are:

Symbol	Meaning
FP_RN	round to nearest
FP_RZ	round toward zero
FP_RM	round toward minus infinity
FP_RP	round toward minus infinity
FP_PX	extended precision
FP_PS	single precision
FP_PD	double precision
FP_X_BSUN	branch/set on unordered exception
FP_X_SNAN	signalling not a number exception
FP_X_OPERR	operand error exception
FP_X_OVFL	overflow exception
FP_X_UNFL	underflow exception
FP_X_DZ	divide by zero exception
FP_X_INEX2	inexact operation exception
FP_X_INEX1	inexact decimal input exception

Note that you should set only one of `FP_RN`, `FP_RZ`, `FP_RM` and `FP_RP`, equally you should set only one of `FP_PX`, `FP_PS` and `FP_PD`.

Note that changing this variable using anything other than a global initialisation will have no effect.

The default value of this variable is `FP_RN|FP_PX`. Note that if you enable any of the exceptions you must install a suitable exception handler.

SEE

fpgetround, fpsetround, fpgetprecision, fpsetprecision, fpgetmask, fpsetmask

_FPCmode — Current math mode

SYNOPSIS

```
#include <ieeefp.h>

int _FPCmode;              maths evaluation package
```

DESCRIPTION

The _FPCmode variable is initialised by the startup code to indicate how floating point maths is performed. The following values are used:

Value	Meaning
1	Co-processor 68881 installed
2	Co-processor 68882 installed
4	68040 installed
-1	I/O based 68881 installed
0	No maths co-processor present

This variable is consulted by the auto-detecting maths routines to decide on how evaluation should proceed.

Your program may wish to interrogate this variable at program startup to ensure that sufficient maths hardware is available for your application, rather than have the user suffer a mysterious crash later on!

fpgetmask, fpsetmask — Get/set exception mask

SYNOPSIS

```
#include <ieeefp.h>

typedef enum fp_except fp_except;

mask = fpgetmask();          get exception mask
mask = fpsetmask(ne);        set exception mask

fp_except mask;              old exception mask
fp_except ne;                new exception mask
```

DESCRIPTION

The fpgetmask and fpsetmask functions manipulate the exception mask of the maths co-processor. fpgetmask returns the current setting for this, whilst fpsetmask changes the exception mask to the value indicated by ne, returning the old value. The values used are:

Symbol	Meaning
FP_X_BSUN	branch/set on unordered
FP_X_SNAN	signalling not a number
FP_X_OPERR	operand error
FP_X_OVFL	overflow
FP_X_UNFL	underflow
FP_X_DZ	divide by zero
FP_X_INEX2	inexact operation
FP_X_INEX1	inexact decimal input

Note that these functions have no effect when the software IEEE emulation is being used, also beware that the standard runtime libraries include no support for dealing with exceptions, hence if you enable any you *must* install a suitable exception handler.

RETURNS

fpgetmask returns the current exception mask. fpsetmask returns the old exception mask.

SEE

_FPCmode, fpgetsticky, fpsetsticky

fpgetprecision, fpsetprecision — Get/set precision

SYNOPSIS

```
#include <ieeefp.h>

typedef enum fp_prec fp_prec;

prec = fpgetround();        get current precision
prec = fpsetround(np);      set new precision

fp_prec prec;               old precision
fp_prec np;                 new precision
```

DESCRIPTION

The `fpgetprecision` and `fpsetprecision` functions manipulate the precision setting of the maths co-processor. `fpgetprecision` returns the current setting for this, whilst `fpsetprecision` changes the precision to the value indicated by `np`, returning the old value. The values used are:

Symbol	Meaning
FP_PX	extended precision
FP_PS	single precision
FP_PD	double precision

Note that these functions have no effect when the software IEEE emulation is being used.

RETURNS

`fpgetround` returns the current precision. `fpsetround` returns the old precision setting.

SEE

`fpgetround, fpsetround`

fpgetround, fpsetround — Get/set rounding mode

SYNOPSIS

```
#include <ieeefp.h>

typedef enum fp_rnd fp_rnd;

rnd = fpgetround();         get current rounding mode
rnd = fpsetround(nr);       set new rounding mode

fp_rnd rnd;                 old rounding mode
fp_rnd nr;                  new rounding mode
```

DESCRIPTION

The `fpgetround` and `fpsetround` functions manipulate the rounding bits of the maths co-processor. `fpgetround` returns the current setting for this, whilst `fpsetround` changes the rounding to the mode indicated by `nr`, returning the old value. The values used for these modes are:

Symbol	Meaning
FP_RN	round to nearest
FP_RZ	round toward zero
FP_RM	round toward minus infinity
FP_RP	round toward minus infinity

Note that these functions have no effect when the software IEEE emulation is being used.

RETURNS

`fpgetround` returns the current rounding mode. `fpsetround` returns the old rounding mode.

SEE

`fpgetprecision, fpsetprecision`

fpgetsticky, fpsetsticky — *Get/set accrued exceptions*

SYNOPSIS

```
#include <ieeefp.h>

typedef enum fp_except fp_except;

prec = fpgetsticky();      get accrued exception byte
prec = fpsetsticky(ne);    set accrued exception byte

fp_except prec;            old accrued exception byte
fp_except ne;              new accrued exception byte
```

DESCRIPTION

The `fpgetsticky` and `fpsetsticky` functions manipulate the accrued exception byte of the maths co-processor. `fpgetsticky` returns the current set of accrued exceptions, whilst `fpsetsticky` changes the current value to `ne`. Note that these functions are 'sticky' because the co-processor never clears any of the bits, hence you may zero the word prior to a calculation and then afterwards collect the total of the problems which occurred. The values used to represent the exception conditions are:

Symbol	Meaning
FP_X_BSUN	branch/set on unordered
FP_X_SNAN	signalling not a number
FP_X_IOP	invalid operation
FP_X_OVFL	overflow
FP_X_UNFL	underflow
FP_X_DZ	divide by zero
FP_X_INEX	inexact operation

Note that these functions have no effect when the software IEEE emulation is being used.

RETURNS

`fpgetsticky` returns the current accrued exception byte. `fpsetsticky` returns the old accrued exception byte.

SEE

`fpgetmask, fpsetmask`

osbind.h

With the advent of the TT and Mega-ST^{E} two new TOSes are available, 2.xx and 3.xx. To cater for these new machines additional OS calls are available in osbind.h.

Bconmap — *Get/Set AUX: device mapping*

SYNOPSIS

```
#include <osbind.h>

val = Bconmap(devno);

long val;                  return value
int devno;                 BIOS device number
```

DESCRIPTION

`Bconmap` is used to control the mapping of the AUX: device (BIOS device 1) which is initially set to the ST compatible serial port. Valid device assignments on the TT are:

devno	Meaning
-2	Return pointer to struct bconmap
-1	Read existing setting
6	Select ST-compatible serial port
7	Select modem 2 (SCC channel B)
8	Select serial 1 (3-wire TT MFP)
9	Select serial 2 (SCC channel A)

Note that these device assignments are *specific* to the TT; other hardware will use different assignments.

RETURNS

As noted above.

SEE

Bconin, Bconout, Bconis, Bconos, Iorec, Rsconf

DMAread — Read sectors from DMA device

SYNOPSIS

```
#include <osbind.h>

err = DMAread(sector, count, buffer, devno);

long err;               error status
long sector;            first sector to read
int count;              number of sectors to read
void *buffer;           pointer to buffer
int devno;              DMA device id
```

DESCRIPTION

DMAread is used to read count sectors from the DMA device given by devno starting at sector into memory. The values of devno are:

devno	Meaning
0-7	ACSI devices 0-7
8-15	SCSI devices 0-7

buffer is a pointer to a suitable memory block. Note that reads from ACSI devices *must* specify a memory block in system RAM. If a transfer to alternative RAM is required it may be possible to use the memory pointed to by _FRB *provided* the _flock system variable is suitably managed.

RETURNS

DMAread returns 0 normally, or a non-zero error code.

DMAwrite — Write sectors from DMA device

SYNOPSIS

```
#include <osbind.h>

err = DMAwrite(sector, count, buffer, devno);

long err;               error status
long sector;            first sector to write
int count;              number of sectors to write
const void *buffer;     pointer to buffer
int devno;              DMA device id
```

DESCRIPTION

DMAwrite is used to write count sectors from the DMA device given by devno starting at sector from memory. The values of devno are:

devno	Meaning
0-7	ACSI devices 0-7
8-15	SCSI devices 0-7

buffer is a pointer to a suitable memory block. Note that writes to ACSI devices *must* specify a memory block in system RAM. If a transfer from alternative RAM is required it may be possible to use the memory pointed to by _FRB *provided* the _flock system variable is suitably managed.

RETURNS

DMAread returns 0 normally, or a non-zero error code.

EgetPalette — Get contiguous entries from TT CLUT

SYNOPSIS

```
#include <osbind.h>

EgetPalette(num, count, palette)

int num;                    initial CLUT entry to modify
int count;                  number of CLUT entries
short *palette;             pointer to new CLUT entries
```

DESCRIPTION

`EgetPalette` copies the contents of a contiguous set of the TT CLUT registers into the array pointed to by `palette`. `num` gives the first entry to copy, whilst `count` gives the number of entries. The CLUT entries are encoded in the following manner:

bits 15-12	bits 11-8 (Red)				bits 7-4 (Green)				bits 3-0 (Blue)			
Unused	R3	R2	R1	R0	G3	G2	G1	G0	B3	B2	B1	B0

`R0` represents the least-significant bit of the red component of the colour, `R3` the most-significant. Similarly, `G0-G3` give the green component and `B0-B3` the blue component. Note that this (and the other TT specific palette calls) do *not* use the ST compatible method of encoding the colour as per `Setpalette`.

RETURNS

None.

SEE

`Setpalette, Setcolor, EsetColor, EsetPalette`

CAVEATS

This function requires that the program is running on a TT. Applications which use this call should check the high word of the `_VDO` cookie for the value 2 before attempting it.

EgetShift — Get current video shift mode

SYNOPSIS

```
#include <osbind.h>

mode = EgetShift()

int mode;                   video shift register value
```

DESCRIPTION

The `EgetShift` call returns the current setting of the TT video shifter. The meaning of the bits within the value are:

Bit 15	Bit 12	Bits 10-8		Bits 3-0
Smear mode	Grey mode	000	ST low	Current colour bank
		001	ST medium	
		010	ST high	
		100	TT medium	
		110	TT high	
		111	TT low	

RETURNS

The call returns the current value of the video shifter.

SEE

`Setscreen, Getrez, EsetSmear, EsetShift, EsetGray, EsetBank`

CAVEATS

This function requires that the program is running on a TT. Applications which use this call should check the high word of the `_VDO` cookie for the value 2 before attempting it.

EsetBank — Get/set colour lookup bank

SYNOPSIS

```
#include <osbind.h>

old = EsetBank(new)

int old;                old active colour bank
int new;                colour bank to select
```

DESCRIPTION

`EsetBank` selects which of the 16 CLUTs (colour lookup table) is active. If the value passed is negative the active CLUT is not changed and the old value is simply returned.

RETURNS

The call returns the old active CLUT number.

SEE

`Setpalette, EsetPalette, EgetPalette`

CAVEATS

This function requires that the program is running on a TT. Applications which use this call should check the high word of the `_VDO` cookie for the value 2 before attempting it.

EsetColor — Get/set colour entry

SYNOPSIS

```
#include <osbind.h>

old = EsetColor(num, color)

int old;                old TT CLUT value
int num;                TT CLUT entry to modify
int color;              new colour value
```

DESCRIPTION

`EsetColor` sets the absolute colour entry `num` to the value given by `color`. The encoding used for `color` is as follows:

bits 15-12	bits 11-8 (Red)				bits 7-4 (Green)				bits 3-0 (Blue)			
Unused	R3	R2	R1	R0	G3	G2	G1	G0	B3	B2	B1	B0

`R0` represents the least-significant bit of the red component of the colour, `R3` the most-significant. Similarly, `G0-G3` give the green component and `B0-B3` the blue component. Note that this (and the other TT specific palette calls) do *not* use the ST compatible method of encoding the colour as per `Setcolor`.

If `color` is negative then the colour is not changed and the old value is returned.

RETURNS

The old value of the TT CLUT entry is returned.

SEE

`Setcolor, EsetPalette, EgetPalette`

CAVEATS

This function requires that the program is running on a TT. Applications which use this call should check the high word of the `_VDO` cookie for the value 2 before attempting it.

EsetGray — Get/Set grey mode

SYNOPSIS

```
#include <osbind.h>

old = EsetGray(new);

int old;                old grey scale mode
int new;                new grey scale mode
```

DESCRIPTION

`EsetGray` is used to read/write the TT video hardware's grey mode bit. When grey mode is set, the bottom eight bits of the palette value are used as one of 256 possible grey levels. `new` is zero to select colour mode, +ve to select grey mode, or -ve to simply return the old setting.

RETURNS

The old grey scale value.

SEE

`EsetShift, EgetShift`

CAVEATS

This function requires that the program is running on a TT. Applications which use this call should check the high word of the _VDO cookie for the value 2 before attempting it.

EsetPalette — Set contiguous entries of TT CLUT

SYNOPSIS

```
#include <osbind.h>

EsetPalette(num, count, palette)

int num;                intial CLUT entry to modify
int count;              number of CLUT entries
short *palette;         pointer to new CLUT entries
```

DESCRIPTION

`EsetPalette` sets the contents of a contiguous set of the TT CLUT registers. `num` gives the first entry to modify, whilst `count` gives the number of entries. `palette` is a pointer to an array of `count` CLUT entries encoded in the following manner:

bits 15-12	bits 11-8 (Red)				bits 7-4 (Green)				bits 3-0 (Blue)			
Unused	R3	R2	R1	R0	G3	G2	G1	G0	B3	B2	B1	B0

`R0` represents the least-significant bit of the red component of the colour, `R3` the most-significant. Similarly, `G0-G3` give the green component and `B0-B3` the blue component. Note that this (and the other TT specific palette calls) do *not* use the ST compatible method of encoding the colour as per `Setpalette`.

RETURNS

None.

SEE

`Setpalette, EsetColor, EgetPalette`

CAVEATS

This function requires that the program is running on a TT. Applications which use this call should check the high word of the _VDO cookie for the value 2 before attempting it.

EsetShift — Set current video shift mode

SYNOPSIS

```
#include <osbind.h>

old = EsetShift(new)

int old;                old shift mode register value
int new;                old shift mode register value
```

DESCRIPTION

The `EsetShift` call is used to change the setting of the TT video shifter. The meaning of the bits within the values are:

Bit 15	Bit 12	Bits 10-8		Bits 3-0
Smear mode	Grey mode	000	ST low	Current colour bank
		001	ST medium	
		010	ST high	
		100	TT medium	
		110	TT high	
		111	TT low	

RETURNS

The call returns the old value of the video shifter.

SEE

`Setscreen, Getrez, EsetSmear, EgetShift`

CAVEATS

This function requires that the program is running on a TT. Applications which use this call should check the high word of the _VDO cookie for the value 2 before attempting it.

EsetSmear — Get/Set smear mode

SYNOPSIS

```
#include <osbind.h>

old = EsetSmear(new);

int old;                old smear mode
int new;                new smear mode
```

DESCRIPTION

`EsetSmear` is used to read/write the TT video hardware's smear mode bit. When smear mode is set, the video hardware displays video pixels with value 0 as the last non-zero colour rather than colour zero itself. This can be used to change the colour of a filled-polygon by only changing its outline rather than via a complete re-fill. `new` is zero to select colour mode, +ve to select grey mode, or -ve to simply return the old setting.

RETURNS

The old smear value.

SEE

`EsetShift, EgetShift`

CAVEATS

This function requires that the program is running on a TT. Applications which use this call should check the high word of the _VDO cookie for the value 2 before attempting it.

Getrez — Find current screen mode

SYNOPSIS

```
#include <osbind.h>

res = Getrez();

int 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)
4	TT medium resolution (640x480x4)
6	TT high resolution (1280x1024x1)
7	TT low resolution (320x480x8)

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 future extended screen modes.

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 or VDI for the information rather than relying on hard-coded constants based on the result of this call.

Maddalt — Inform GEMDOS of alternative memory

SYNOPSIS

```
#include <osbind.h>

err = Maddalt(start, size);

int err;                    0 or error code
void *start;                start of memory block
long size;                  size of memory block
```

DESCRIPTION

This call is used to inform GEMDOS of the presence of alternative RAM; it should not be needed unless you have added custom memory to the system which is not detected by the BIOS at boot time. `start` is pointer to the block of memory, whilst `size` gives the number of blocks in the block.

RETURNS

`Maddalt` returns 0 for success, or a GEMDOS error code.

SEE

`Mxalloc`

Mxalloc — Allocate block of from preferred pool

SYNOPSIS

```
#include <osbind.h>

base = Mxalloc(amount, mode);

void *base;                 base of block allocated
long amount;                amount of memory requested
int mode;                   memory type preference
```

DESCRIPTION

The `Mxalloc` function is used to obtain blocks of memory from a preferred 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. The `mode` parameter gives the type of memory the application is interested in receiving and has the values:

Mode	Meaning
0	ST RAM only
1	alternative RAM only
2	either, ST RAM preferred
3	either, alternative preferred.

To determine the size of the largest free block in the system of a given type, 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

`Mxalloc` 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

This call was added in TOS 2.0. An application may detect the presence of alternative memory (aka fast memory) by interrogating the `_FRB` cookie, which will only be present on machines with alternative RAM.

NVMaccess — Read/Write non-volatile memory

SYNOPSIS

```
#include <osbind.h>

err = NVMaccess(op, start, count, buffer);

int err;                    error status
int op;                     operation to perform
int start;                  first location to read/write
int count;                  number of bytes to read/write
char *buffer;               pointer to buffer
```

DESCRIPTION

`NVMaccess` is used to access the non-volatile memory in the Atari TT's real time clock. `op` gives the operation which is to be performed and has the following values:

op	Meaning
0	Read NVM data
1	Write NVM data
2	Reset and initialise NVM

`start` gives the first of `count` bytes which should be read/written from/to `buffer`.

RETURNS

`NVMaccess` returns 0 normally, or a non-zero error code.

CAVEATS

At the time of writing no public entries within the non-volatile memory have been committed by Atari. *All* locations are reserved for use by Atari.

Pexec — Create/Execute process

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 NUL 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.
7	Create a basepage. For this mode `path` holds the program load flags as set in the executable file's load bits. `tail` and `env` have their normal meanings. The value returned is the address of the base page created. 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. `Pexec` mode 7 is only available on GEMDOS version 0.24 (TOS 2.0) and above.

stdlib.h

getopt — Get option letter from argument vector

SYNOPSIS

```
#include <stdlib.h>

c = getopt(argc,argv,optstring);

int c;                      argument character
int argc;                   argument count
const char *argv[];         argument vector
const char *optstring;      string containing valid opts

extern char *optarg;        pointer to option argument
extern int optind;          index of next argument
extern int opterr;          error message setting
```

DESCRIPTION

The `getopt` function returns the next option letter in `argv` which matches a letter in `optstring`. `optstring` contains all the option letters which are to be recognised, optionally followed by a colon (`:`) when an argument is required by the option. Such an argument may either be concatenated with the option letter, or be the next argument. The external variable `optarg` is set to point to any such argument. If the colon is doubled (`::`) then the argument is optional and if present *must* be concatenated to the option letter, if no argument was present then `optarg` will be `NULL`.

The external variable `optind` is used to track the next `argv` index which `getopt` will use and is normally initialised to 1 by the first call to `getopt`.

When all options have been processed (i.e. the first argument which does not start with a '-'), or the special delimiter '--' has been encountered the value -1 is returned and and the '--' argument skipped.

When an unrecognised option is encountered, or an argument option is omitted where one was expected, an error message is printed on `stderr` and the value '?' returned. The printing of error messages may be disabled by setting the external variable `opterr` to 0.

Note that unlike `argopt`, `getopt` does *not* recognise a '/' as an option prefix.

RETURNS

The value of the character obtained as an option, '?' for an invalid option or -1 if no more arguments are available.

SEE

argopt, main

EXAMPLE

```
/*
 * parse the command lines:
 *   myprog -x -ypdq -z -g moo blah
 */

#include <stdlib.h>

int main(int argc, char *argv[])
{
   int c;
   char *file,*status = NULL;
   int x=0,z=0;

   while ((c=getopt(argc,argv,"xy::zg:"))!=-1)
      switch (c) {
         case 'x':
            x++;
            break;

         case 'z':
            z++;
            break;

         case 'y':
            if (optarg)
               status=optarg;
            break;

         case 'g':
            file=optarg;
            break;
```

```
        case '?':
          abort();
          break;
      }

   for (; optind<argc; optind++)
     process(argv[optind],x,z,status,file);

   return 0;
}
```

spawn — Launch new process

SYNOPSIS

```
#include <stdlib.h>

error = spawnl(mode,prog,arg0,…,argn,NULL);
error = spawnv(mode,prog,argv);

error = spawnle(mode,prog,arg0,…,argn,NULL,envp);
error = spawnve(mode,prog,argv,envp);

error = spawnlp(mode,prog,arg0,…,argn,NULL);
error = spawnvp(mode,prog,argv);

error = spawnlpe(mode,prog,arg0,…,argn,NULL,envp);
error = spawnvpe(mode,prog,argv,envp);

int error;                  error code
int mode;                   execution mode of program
const char *prog;           program name
const char *arg0;           argument #0
const char *argn;           argument #n
const char *argv[];         argument vector
const char *envp[];         environment pointers

extern int _aecl;           Extended command lines flag
```

DESCRIPTION

The spawn... family of functions provide the most control over the creation of processes. The mode parameter specifies how the named program is to be launched.

The values of this are:

Symbol	Meaning
P_WAIT	Spawn process and wait for child to terminate, returning termination code.
P_NOWAIT	Spawn process concurrently, termination code available from wait. This is equivalent to fork...
P_OVERLAY	Overlay current process. Does not return. This is equivalent to exec...

Note that P_NOWAIT is not supported by GEMDOS, so the calling process actually waits for the child to terminate before continuing. Also the P_OVERLAY mode is not available from GEMDOS and is simulated by the spawn... family function terminating via _exit after execution; this mode is designed for use with vfork.

Details on the various modes of this command are as under the fork... family of functions.

RETURNS

The return code is as specified above for a successful execution, otherwise the specified program file cannot be found, a -1 return is made, and additional error information can be found in errno and _OSERR.

SEE

Pexec, _exit, exec..., fork..., vfork, wait

string.h

string.h is the ANSI string definitions file. To support other pre-ANSI platforms some additional string and memory functions are available.

bcmp, bcopy, bzero — BSD memory block operations

SYNOPSIS

```
#include <string.h>

x = bcmp(a,b,n);          Compare two memory blocks
s = bcopy(from,to,n);     Copy a memory block
s = bzero(to,n);          Zero a memory block

void *to;                 destination pointer
const void *from;         source pointer
size_t n;                 number of bytes
const void *a,*b;         block pointers
void *s;                  return pointer
int x;                    return value
```

DESCRIPTION

These functions manipulate blocks of memory in various ways. The `bcmp` function is identical to the `memcmp` function and compares two blocks of memory. The `bzero` function zeroes the nominated block of memory; note that unlike the `memset` function it is not possible to change the value which is used for the initialisation. The `bcopy` function copies the area from one location to the other in the same manner as the ANSI `memcpy` function; note that `from` and `to` are transposed with respect to `memcpy`.

RETURNS

As noted above.

SEE

`memcmp, memcpy, memset`

index, rindex — Find character

SYNOPSIS

```
#include <string.h>

p = index(s,c);           find character in string
p = rindex(s,c);          find last character in string

char *p;                  updated string pointer
const char *s;            input string pointer
int c;                    character to be located
```

DESCRIPTION

The `index` function scans the input string to find the first occurrence of the character specified by argument `c`. The `rindex` function scans the input string to find the last occurrence of the character specified by argument `c`.

Note that these functions are almost identical to the ANSI `strchr` and `strrchr` functions, however these functions *never* match the trailing `\0`.

RETURNS

A `NULL` pointer is returned if the input string is empty or if the specified character is not found.

sys/stat.h

fstat — Get status of file handle

SYNOPSIS

```
#include <sys/types.h>
#include <sys/stat.h>

ret = fstat(fd,statbuf);

int ret;                    0 if successful
int fd;                     file handle of file
struct stat *statbuf;       stores information about file
```

DESCRIPTION

The `fstat` function returns UNIX-style file status information about the file specified by `fd`. The buffer returned is defined in sys/stat.h as follows:

```
struct stat {
   dev_t st_dev;              disk drive number
   ino_t st_ino;              inode number (not used)
   unsigned short st_mode;    file mode flags
   short st_nlink;            number of links (always 1)
   short st_uid;              user id (not used)
   short st_gid;              group id (not used)
   dev_t st_rdev;             same as st_dev
   off_t st_size;             file size in bytes
   time_t st_atime;           time of last access
   time_t st_mtime;           time of last modification
   time_t st_ctime;           time of creation
};
```

Note that the header file sys/types.h must be included prior to sys/stat.h as this defines the types `dev_t`, `ino_t`, `dev_t` and `off_t`.

RETURNS

On success, the `fstat` function returns 0.

umask — Get/set file creation mask

SYNOPSIS

```
#include <fcntl.h>

old = umask(mask)

int old;                    existing mask setting
int mask;                   desired setting for file mask
```

DESCRIPTION

`umask` sets the file mode creation mask to its argument and returns the previous value of the mask. The default value is 0 (all permissions available). In practice this function is only useful when called as `umask(S_IWRITE)` to create read only files by default, since that is the only UNIX® style protection bit implemented by GEMDOS.

RETURNS

The old value of the file mask is returned

SEE

`chmod, creat, Fcreat, open`

time.h

stime ***Set current system time***

SYNOPSIS

```
#include <time.h>

err = stime(time)

int err;                    error return
const time_t *time;         pointer to time
```

DESCRIPTION

The `stime` function changes the current time and date in the system clock to the value pointed to by `time`.

RETURNS

The function returns 0 on successfully changing the time, or -1 to indicate an error, with further information in `errno`.

SEE

`Tsetdate, Tsettime, time, utime`

unistd.h

exec ***Overlay current process***

SYNOPSIS

```
#include <unistd.h>

error = execl(prog,arg0,arg1,…,argn,NULL);
error = execv(prog,argv);

error = execle(prog,arg0,arg1,…,argn,NULL,envp);
error = execve(prog,argv,envp);

error = execlp(prog,arg0,arg1,…,argn,NULL);

error = execvp(prog,argv);

error = execlpe(prog,arg0,arg1,…,argn,NULL,envp);
error = execvpe(prog,argv,envp);

int error;                  error code
const char *prog;           program name
const char *arg0;           argument #0
const char *arg1;           argument #1
const char *argn;           argument #n
const char *argv[];         argument vector
const char *envp[];         environment pointers

extern int _aecl;           extended command lines flag
```

DESCRIPTION

These functions mimic the UNIX® process overlay commands. GEMDOS does not actually support this, so the implementation spawns a child, terminating (via `_exit`) on return. This family of functions are designed for use with the `vfork` function.

Details on the various modes of this command are as under the `fork...` family of functions.

RETURNS

If the call succeeds the function never returns. If the specified program file cannot be found, a -1 return is made, and additional error information can be found in `errno` and `_OSERR`. Note that you must call the `wait` function in order to obtain the completion code from the child process.

SEE

`Pexec, _exit, fork, spawn, vfork, wait`

vfork — *Spawn new process*

SYNOPSIS

```
#include <unistd.h>

pid = vfork();

int pid;                    process id of child or zero
```

DESCRIPTION

`vfork` is used to create new processes. Under UNIX `fork` creates a new concurrent process, `vfork` simulates this by 'borrowing' the parents address space until a call to `exec...` is made.

`vfork` returns 0 in the child's context and (later) the process id of the child in the parent's context.

When using `vfork` you must be careful that you do not alter the context of the parent process in the child. The child process must not return from the function which called `vfork`, if this were to happen the parent process would then return to a stack frame which had been deallocated by the child.

If the call to `exec...` should fail then the child *must* call `_exit`, rather than `exit` which would otherwise close the standard I/O files.

RETURNS

`vfork` returns 0 in the child's context and (later) the process id of the child in the parent's context.

SEE

`exec..., fork..., spawn..., wait`

EXAMPLE

```
#include <unistd.h>
#include <stddef.h>

int main(int argc,char *argv[])
{
   if (!vfork())
      execl(argv[1],argv[1],NULL);
   else
      execl(argv[2],argv[2]),NULL);
}
```

vdi.h

vdi.h includes all the binding routines which are available for call the GEM VDI. There are many new functions available for use either with GDOS or FSM-GDOS.

v_pgcount — *Set number of copies for laser printer*

SYNOPSIS

```
#include <vdi.h>

v_pgcount(handle,n);

int handle;                 device handle
int n;                      number of additional copies
```

DESCRIPTION

`v_pgcount` is used to change the number of copies generated by laser printer driver from the default of 1. The parameter, `n`, gives the number of *additional* copies which are required.

This function requires that GDOS, Font-GDOS or FSM-GDOS is loaded.

RETURNS

None.

SEE

v_clear_disp_list, v_clrwk, v_opnwk

vq_extnd — Extended Inquire

SYNOPSIS

```
#include <vdi.h>

vq_extnd(handle,flag,work_out);

int handle;              workstation handle
int flag;                0=normal; 1= extended inquire
short *work_out;         values returned
```

DESCRIPTION

This function can be used to return the information returned by the v_opnwk or v_opnvwk calls (if flag==0) or additional values if flag==1. The work_out array must have room for at least 57 shorts. The values returned when flag=0 are detailed under v_opnwk. The values returned when flag==1 are as follows:

work_out[0]	Type of screen: 0 = not screen. 4 = 'normal' screen with common graphics and character memory. Other values are not applicable to the ST.
work_out[1]	Number of background colours available.
work_out[2]	Text effects supported. See vst_effects.
work_out[3]	Scaling of rasters: 0 = scaling not supported. 1 = scaling supported.
work_out[4]	Number of planes available.
work_out[5]	Lookup table supported 0 = table supported. 1 = table not supported.

work_out[6]	Performance factor. Number of 16x16 pixel raster operations per second.
work_out[7]	Contour fill capability: 0 = no. 1 = yes.
work_out[8]	Character rotation ability: 0 = none. 1 = multiples of 90 degrees only. 2 = any angle.
work_out[9]	Number of writing mode available.
work_out[10]	Highest level of input mode available: 0 = none. 1 = request. 2 = sample.
work_out[11]	Text alignment capability flag: 0 = no. 1 = yes.
work_out[12]	Inking capability flag: 0 = no. 1 = yes.
work_out[13]	Rubber-banding capability flag: 0 = no. 1 = rubber-band lines possible. 2 = rubber-band lines and rectangles possible.
work_out[14]	Maximum vertices for polyline, polymarker or filled area (-1 = no maximum).
work_out[15]	Maximum index for intin (-1 = no maximum).
work_out[16]	Number of keys on the mouse.
work_out[17]	Styles available for wide lines: 0 = no. 1 = yes.
work_out[18]	Writing modes available for wide lines: 0 = no. 1 = yes.
work_out[19]	Clipping enabled flag: 0 = no. 1 = yes.
work_out[20] ... work_out[44]	Reserved.

work_out[45]	First X co-ordinate of clipping rectangle
work_out[46]	First Y co-ordinate of clipping rectangle
work_out[47]	Second X co-ordinate of clipping rectangle
work_out[48]	Second Y co-ordinate of clipping rectangle
work_out[49]	Reserved.
...	
work_out[56]	

SEE

v_opnwk, v_opnvwk

v_bez — Draw bezier curve

SYNOPSIS

```
#include <vdi.h>

v_bez(handle,count,xy,bezarr,extent,totpts,totmoves);

int handle;                workstation handle
int count;                 number of vertices
const short *xy;           array of vertices
const char *bezarr;        array of vertex descriptors
short extent[4];           extent of resultant bezier
short *totpts;             total points in polygon
short *totmoves;           total moves in polygon
```

DESCRIPTION

v_bez is used to draw an unfilled bezier curve on the device referenced by handle. xy points to a set of (x, y) co-ordinate pairs, whilst bezarr lists the attributes of each of the corresponding points. The following bits are used in bezarr:

Bit	Value	Meaning
0	0	Point begins a polyline section.
0	1	Point is the first point of a set of 4 bezier points in the sequence: first anchor point, first control point, second control point, second anchor point.
1	1	Point is a jump point; the current point is moved to this point without a joining line

The total number of points in the resulting polygon is returned in totpts, whilst the total number of moves in the resulting polygon is returned in totmoves. The bounding box of the polygon is returned in extent.

This function requires that Font-GDOS or FSM-GDOS is loaded.

SEE

v_pline, v_bez_fill

EXAMPLE

```
#include <aes.h>
#include <vdi.h>
#include <osbind.h>

short work_out[57];
short work_in[11]={0,1,1,1,1,1,1,1,1,1,2};

main()
{
   short h, junk;
   short pts[8],extent[4],totpts,totmoves,act;

   appl_init();
   work_in[0] = Getrez()+ 2;
   h = graf_handle(&junk, &junk, &junk, &junk);
   v_opnvwk(work_in, &h, work_out);
   v_bez_on(h);
   v_bez_qual(h, 100, &act);

   pts[0]=100;
   pts[1]=100;

   pts[2]=100;
   pts[3]=400;

   pts[4]=400;
   pts[5]=100;

   pts[6]=400;
   pts[7]=400;

   v_bez(h, 4, pts, "\1\0\0\0", extent, &totpts,
    &totmoves);
```

```
    v_bez_off(h);
    v_clsvwk(h);
    appl_exit();
}
```

v_bez_con — Control GDOS bezier facilities

SYNOPSIS

```
#include <vdi.h>

qual = v_bez_con(handle, onoff);

int qual;                maximum bezier depth
int handle;              workstation handle
int onoff;               1 to enable, 0 to disable
```

DESCRIPTION

`v_bez_con` is used to enable or disable GDOS bezier capabilities. The `onoff` parameter is 0 to disable bezier facilities or 1 to enable them. Note that two macros are provided for this purpose, `v_bez_on()` and `v_bez_off()`.

Note that failure to disable bezier facilities prior to closing the (virtual) workstation may cause the VDI to crash.

This function requires that Font-GDOS or FSM-GDOS is loaded.

RETURNS

When enabling bezier operation a value, ranging from 0 to 7, is returned giving the maximum bezier depth, with 2^{qual} giving the number of line segments used to make up the curve.

SEE

`v_set_app_buff, v_bez, v_bezfill`

v_bez_fill — Draw filled bezier curve

SYNOPSIS

```
#include <vdi.h>

v_bez_fill(handle,count,xy,bezarr,extent,totpts,
                          totmoves);

int handle;              workstation handle
int count;               number of vertices
const short *xy;         array of vertices
const char *bezarr;      array of vertex descriptors
short extent[4];         extent of resultant bezier
short *totpts;           total points in polygon
short *totmoves;         total moves in polygon
```

DESCRIPTION

`v_bez_fill` is used to draw a filled bezier curve on the device referenced by `handle`. `xy` points to a set of (`x`, `y`) co-ordinate pairs, whilst `bezarr` lists the attributes of each of the corresponding points. The following bits are used in `bezarr`:

Bit	Value	Meaning
0	0	Point begins a polyline section.
0	1	Point is the first point of a set of 4 bezier points in the sequence: first anchor point, first control point, second control point, second anchor point.
1	1	Point is a jump point; the current point is moved to this point without a joining line

The total number of points in the resulting polygon is returned in `totpts`, whilst the total number of moves in the resulting polygon is returned in `totmoves`. The bounding box of the polygon is returned in `extent`.

This function requires that Font-GDOS or FSM-GDOS is loaded.

SEE

`v_pline, v_bez`

v_bez_qual — Set bezier quality

SYNOPSIS

```
#include <vdi.h>

v_bez_qual(handle, percent, actual);

int handle;             workstation handle
int percent;            speed/quality percentage
short *actual;          actual percentage selected
```

DESCRIPTION

`v_bez_qual` sets the bezier speed/quality trade-off parameter as a percentage of the quality (hence 100% is best quality but slowest). The quality factor is passed in `percent` as a value from 0 to 100. The value selected by GDOS (approximated to the 8 levels available) is returned in actual.

This function requires that Font-GDOS or FSM-GDOS is loaded.

SEE

`v_bez, v_bez_fill, v_set_app_buff`

v_flushcache — Flush FSM font cache

SYNOPSIS

```
#include <vdi.h>

v_flushcache(handle);

int handle;             workstation handle
```

DESCRIPTION

The `v_flushcache` call empties the FSM font caches. This function requires that FSM GDOS is loaded.

SEE

`v_loadcache, v_savecache`

v_ftext — Draw graphics text

SYNOPSIS

```
#include <vdi.h>

v_ftext(handle,x,y,str);

int handle;             workstation handle
int x;                  x co-ordinate of start
int y;                  y co-ordinate of start
const char *str;        characters to output
```

DESCRIPTION

This function is identical to the normal VDI `v_gtext` function, however the text drawn takes into account the remainder values from `vqt_advance`, resulting in more accurate spacing.

v_getoutline — Get FSM outline

SYNOPSIS

```
#include <vdi.h>

v_getoutline(handle, ch, component);

int handle;             workstation handle
int ch;                 character to obtain outline
fsm_component_t *cpt;   FSM component pointer
```

DESCRIPTION

This function requires FSM-GDOS for operation and obtains a pointer to the outline for the character `ch`.

A pointer to the character outline is returned in `component`. A description of the `fsm_component_t` data structure is beyond the scope of this document.

This function requires that FSM-GDOS is loaded.

SEE

`v_killoutline`

v_killoutline — Kill FSM outline

SYNOPSIS

```
#include <vdi.h>

v_killoutline(handle, component);

int handle;                 workstation handle
fsm_component_t *cpt;       FSM component pointer
```

DESCRIPTION

This function requires FSM-GDOS for operation and releases the memory occupied by the FSM component (obtained via `v_getoutline`).

This function requires that FSM-GDOS is loaded.

SEE

`v_getoutline`

v_loadcache — Load FSM font cache from disk

SYNOPSIS

```
#include <vdi.h>

err = v_loadcache(handle,name,mode);

int err;                    non-zero if an error occurred
int handle;                 workstation handle
char *name;                 file name
int mode;                   0 to append, 1 to replace
```

DESCRIPTION

`v_loadcache` is used to restore a font cache previously saved using `v_savecache`. `name` specifies the file which is to be loaded, normally this should have a .FSM extension. `mode` specifies whether the file replaces or appends to the existing font cache. If the value is 0 the file is appended, otherwise the font cache is replaced.

This function requires that FSM-GDOS is loaded.

SEE

`v_savecache, v_flushcache`

v_savecache — Save FSM font cache to disk

SYNOPSIS

```
#include <vdi.h>

err = v_savecache(handle,name);

int err;                    non-zero if an error occurred
int handle;                 workstation handle
char *name;                 file name to save cache in
```

DESCRIPTION

`v_savecache` is used to save the font cache for reloading by `v_loadcache`. `name` specifies the file which in which the cache is to be saved, normally this should have a .FSM extension.

This function requires that FSM-GDOS is loaded.

SEE

`v_loadcache, v_flushcache`

v_set_app_buff — Reserve bezier workspace

SYNOPSIS

```
#include <vdi.h>

v_set_app_buff(buffer, npara);

void *buffer;               pointer to buffer
int npara;                  number of paragraphs
```

DESCRIPTION

This call makes the nominated memory block available for use by the GDOS bezier extensions. If this call is not made, a default 8K buffer is allocated by GDOS.

`buffer` is a pointer to the memory block, and `npara` is the number of *paragraphs* available in the block (a paragraph is 16 bytes of memory).

Note that *no workstation handle* is passed. This function requires that Font-GDOS or FSM-GDOS is loaded.

SEE

`v_bez_con`

vq_vgdos — Obtain GDOS version number

SYNOPSIS

```
#include <vdi.h>

ver = vq_vgdos();

unsigned long ver;        GDOS revision marker
```

DESCRIPTION

This function indicates what version of (or whether) GDOS is loaded. GDOS is the part of GEM that was left out of the ST's ROMs; it provides the ability to load fonts from disk, load printer drivers and use device-independent co-ordinates.

The values returned by the function are:

Symbol	Meaning
`GDOS_FNT`	FONT GDOS is loaded. This is the same as FSM GDOS with the font scaling module removed.
`GDOS_FSM`	FSM GDOS is loaded. This indicates that all FSM functions are available.
`GDOS_NONE`	No GDOS is present. Note that there is no code to indicate a pre-FSM release of GDOS, this is indicated by any other value.

You should always use this function to determine whether GDOS is loaded, otherwise the system will crash if you use a facility not provided by the ROM (such as opening a physical workstation).

SEE

`v_opnwk, vq_gdos`

vqt_advance — Inquire FSM advance vector

SYNOPSIS

```
#include <vdi.h>

vqt_advance(handle, ch, advx, advy, xrem, yrem);

int handle;               workstation handle
int ch;                   character
short *advx;              X advance value
short *advy;              X advance value
short *xrem;              X remainder (modulo 16384)
short *yrem;              Y remainder (modulo 16384)
```

DESCRIPTION

This function returns the X and Y advance values for the character `ch` in `advx` and `advy` respectively, together with any fractional advances in `xrem` and `yrem`. Note that this function takes into account the current rotation, specified by `vst_rotation`.

This function requires that FSM-GDOS is loaded.

SEE

`vst_rotation`

vqt_cachesize — Inquire FSM font cache size

SYNOPSIS

```
#include <vdi.h>

vqt_cachesize(handle,which,size);

int handle;              workstation handle
int which;               cache to obtain size of
long *size;              size of selected cache
```

DESCRIPTION

`vqt_cachesize` obtains the size of one of the FSM caches. The size of the cache required is dictated by the `which` parameter; this is 0 to find the size of the largest space in the bitmap cache, or 1 to find the size of the largest space in the data structure cache. The size of the selected cache is returned in `size`.

This function requires that FSM-GDOS is loaded.

SEE

`v_flushcache, v_loadcache, v_savecache`

vqt_devinfo — Inquire device status information

SYNOPSIS

```
#include <vdi.h>

vqt_devinfo(handle,devnum,exists,name);

int handle;              workstation handle
int devnum;              device number to investigate
short *exists;           non-zero if device exists
char *name;              name of device (if present)
```

DESCRIPTION

`vqt_devinfo` is used to ascertain whether a particular driver ID has been installed and what driver is associated with it. `handle` contains a workstation handle for the device. On return `exists` is non-zero if the device exists, whilst `name` contains the ASCII name for the device.

This function requires that Font-GDOS or FSM-GDOS is loaded.

SEE

`v_opnwk, v_opnvwk`

EXAMPLE

```
/*
 * Obtain all valid workstation IDs and print out the
 * name of their driver
 */

#include <aes.h>
#include <vdi.h>

#define MAX_SCREEN_ID 10
#define MAX_WORKSTATION_ID 128

int main(void)
{
   short h,exists;
   char name[128];
   int i,dev;
   short work_in[11], work_out[57];
```

```
    appl_init();
    for (dev = 1; dev <= MAX_WORKSTATION_ID; dev++) {
       for (i=1; i<10; i++)
          work_in[i]=1;
       work_in[0] = dev;
       work_in[10] = 2;
       if (dev <= MAX_SCREEN_ID) {
          h=graf_handle(&h,&h,&h,&h);
          v_opnvwk(work_in,&h,work_out);
       }
       else
          v_opnwk(work_in,&h,work_out);
       if (h) {
          vqt_devinfo(h,dev,&exists,name);
          if (exists)
            printf("ID=%d, \"%s\"\n",dev,name);
          if (dev<=MAX_SCREEN_ID)
            v_clsvwk(h);
          else
            v_clswk(h);
       }
    }
    return appl_exit();
}
```


SYNOPSIS

```
#include <vdi.h>

vqt_f_extent(handle,str,pts);

int handle;              workstation handle
const char *str          string to find size of
short *pts;              values returned
```

DESCRIPTION

This function returns the screen area needed to display a string of graphics text using the current text attributes, taking *into account* any remainders indicated by `vqt_advance`. This gives how much screen area will be used if `v_ftext` is used to display that string. The diagram below shows how the points that mark the boundary of the string are numbered:

The `pts` array, which should be large enough to hold 8 shorts will be returned as follows:

`pts[0]`	x co-ordinate of point 1.
`pts[1]`	y co-ordinate of point 1.
`pts[2]`	x co-ordinate of point 2.
`pts[3]`	y co-ordinate of point 2.
`pts[4]`	x co-ordinate of point 3.
`pts[5]`	y co-ordinate of point 3.
`pts[6]`	x co-ordinate of point 4.
`pts[7]`	y co-ordinate of point 4.

This function requires that FSM-GDOS is loaded.

SEE

v_ftext, vqt_advance

vqt_f_name — Return font name and index

SYNOPSIS

```
#include <vdi.h>

index=vqt_f_name(handle, num, name, isfsm);

int index;          the font index
int handle;         workstation handle
int num;            font number
char name[32];      font name
short *isfsm;       non-zero if FSM outline font
```

DESCRIPTION

This function returns the name of a font and its font index. The function that changes the current font, `vst_font`, requires a font index which should be obtained using `vqt_f_name`.

The font numbers that are passed in the `num` parameter start at 1 and are followed by 2, 3, etc until the number of loaded fonts. The number of loaded fonts is returned by the `vst_load_fonts` call. Font number 1 is the system font.

The `name` parameter must point to a buffer of at least 32 characters which will be filled in to give the font name.

The `isfsm` flag is set to 1 by the binding to indicate that the selected font is an FSM outline font. If the font is a bitmap font then 0 is returned.

This function requires that Font-GDOS or FSM-GDOS is loaded.

RETURNS

This function returns the font index.

SEE

vqt_name, vqt_f_extent

vqt_get_tables — Obtain pointer to GASCII tables

SYNOPSIS

```
#include <vdi.h>

vqt_get_tables(handle,gascii,style);

int handle;             workstation handle
short **gascii;         pointer to GASCII table
short **style;          pointer to style table
```

DESCRIPTION

`vqt_get_tables` returns the addresses of two tables, which are used internally by FSM-GDOS to translate ASCII characters passed to `v_gtext` into a short word which is recognised by the FSM character generator. These translations are contained in the `gascii` table. This table contains 223 entries with the first entry giving the translation for character 32, the second for character 33....etc.

The second table, `style`, gives information on which font file the character was generated from. When a character is generated the FSM module selects either the main, symbol or Hebrew font based on this table, the values of which are:

Value	Meaning
0	From main font file
1	From symbol font file
2	From Hebrew font file

These tables may then be used by the application to access fonts available in the FSM character sets by modifying the values stored there.

This function requires that FSM-GDOS is loaded.

SEE

v_gtext

vst_arbpt — Select arbitrary point size

SYNOPSIS

```
#include <vdi.h>

set = vst_arbpt(handle,point,ch_w,ch_h,cell_w,cell_h);

int set;                point size selected
int handle;             workstation handle
int point;              requested point size
short *ch_w;            character width
short *ch_h;            character height
short *cell_w;          cell width
short *cell_h;          cell height
```

DESCRIPTION

This function selects an arbitrary point size for an FSM font. This differs from the `vst_point` call which will only allow the sizes mentioned in the EXTEND.SYS file to be selected.

This function requires that FSM-GDOS is loaded.

SEE

`vst_point`

vst_error — Set FSM error mode

SYNOPSIS

```
#include <vdi.h>

vst_error(handle, mode, err);

int handle;             workstation handle
int mode;               error mode
short *err;             pointer to error variable
```

DESCRIPTION

`vst_error` configures the way in which FSM errors are reported. The default, `mode==1`, places FSM error messages on the screen. If `vst_error` is called with `mode==0` then future errors are placed in the `err` variable. The following error codes are used:

Value	Meaning
0	No error
1	Character not found in font
8	Error reading file
9	Error opening file
10	Bad file format
11	Out of memory/cache full
-1	Miscellaneous error

An FSM error may be generated by any of the following calls:

```
v_ftext(), v_gtext(), v_justified(), v_opnvwk(),
v_opnwk(), vqt_advance(), vqt_extent(), vqt_f_extent(),
vqt_f_name(), vqt_fontinfo(), vqt_name(), vqt_width(),
vst_arbpt(), vst_font(), vst_height(), vst_load_fonts(),
vst_point(), vst_setsize(), vst_unload_fonts()
```

This function requires that Font-GDOS or FSM-GDOS is loaded.

vst_scratch — Set FSM scratch allocation mode

SYNOPSIS

```
#include <vdi.h>

vst_scratch(handle, mode);

int handle;              workstation handle
int mode;                scratch buffer mode
```

DESCRIPTION

`vst_scratch` sets the manner in which the size of the VDI effects buffer (used by effects such as bold, italic etc.) is calculated. The `mode` parameter should have one of the following values:

Value	Meaning
0	Account for effects on FSM fonts when calculating buffer size
1	Ignore FSM fonts when calculating buffer size
2	Allocate no scratch buffer (hence making any effect use illegal)

This function requires that Font-GDOS or FSM-GDOS is loaded.

SEE

`vst_effects`

vst_setsize — Set cell width to arbitrary point size

SYNOPSIS

```
#include <vdi.h>

set =
vst_setsize(handle,point,ch_w,ch_h,cell_w,cell_h);

int set;                 point size selected
int handle;              workstation handle
int point;               requested point size
short *ch_w;             character width
short *ch_h;             character height
short *cell_w;           cell width
short *cell_h;           cell height
```

DESCRIPTION

`vst_setsize` sets the graphic text character width in points. This allows an arbitrary set size to be used for the character width. Note that the next call to `vst_point`, `vst_arbpt` or `vst_height` will override any set size set by this call. This function requires that FSM-GDOS is loaded.

SEE

`vst_arbpt, vst_height, vst_point`

vst_skew — *Set FSM font skewing angle*

SYNOPSIS

```
#include <vdi.h>

set = vst_skew(handle,skew);

int set;                    skew value selected
int handle;                 workstation handle
int skew                    angle of skew required
```

DESCRIPTION

`vst_skew` sets the skew used when generating characters, note that this is independent of the skewing generated using `vst_effects`. The `skew` value (between -900 and 900) represents the number of 10^{ths} of a degree by which the characters are to be skewed. Negative values produce skews to the left, positive values skews to the right. This function requires that FSM-GDOS is loaded.

RETURNS

The function returns the skew value actually set.

SEE

`vst_effects`