SEE

vs_color

EXAMPLE

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

int  main(void)
{
  short  work_in[11]={1,1,1,1,1,1,1,1,1,1,2};
  short  work_out[57];
  short  handle;   /* virtual workstation handle */
  short  junk;
  short  pts[6]={10,20,100,40,20,100};

  appl_init();
  handle=graf_handle(&junk,&junk,&junk,&junk);
  v_opnvwk(work_in,&handle,work_out);
  if (handle)
  {
    vsf_color(handle,GREEN);
    vsf_interior(handle,FIS_USER);
    vsf_style(handle,1);    /* Atari logo */
    v_fillarea(handle,3,pts);
    /* draws a green triangle with corners at
       (10,20), (100,40) and (20,100) */
    v_clsvwk(handle);
  }
  return  appl_exit();
}
```

vsf_interior, vsf_style

Set the fill style

Class: VDI *Category: Fill Area Attributes*

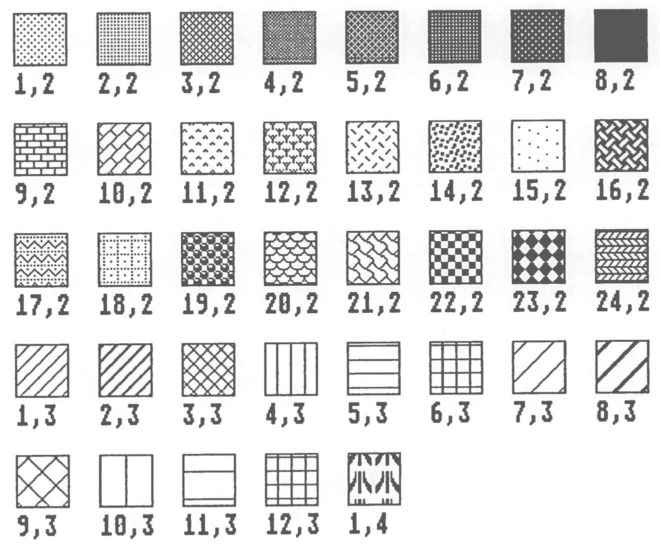
SYNOPSIS

```
#include  <vdi.h>

new_interior=vsf_interior(handle,interior);
new_index=vsf_style(handle,index);

int  new_interior;        the new interior style set
int  new_index;           the new style index set
int  handle;              workstation handle
int  interior;            interior style
int  index;               style index
```

DESCRIPTION

These functions change how the areas that are filled using v_fillarea and other functions that use the fill area attributes, are displayed.

The table below shows the effect of using different style indices. The first number is the index parameter as passed to vsf_style, the second is the interior parameter for vsf_interior:



You *must* first set the style using vsf_interior and then the index using vsf_style. Valid values for the interior parameter are as follows:

FIS_HOLLOW	Hollow interior, set to colour 0.
FIS_SOLID	Solid interior, with colour as set by vsf_color.
FIS_PATTERN	Patterns, as noted above.
FIS_HATCH	Hatches, as noted above.
FIS_USER	User-defined style as set with vsf_udpat. This is an Atari logo by default.

RETURNS

The vsf_interior function returns the style set and vsf_style returns the new index set.

SEE

vsf_udpat, vsf_perimeter, v_fillarea

vsf_perimeter

Set the fill area perimeter visibility

Class: VDI *Category: Fill Area Attributes*

SYNOPSIS

```
#include  <vdi.h>

new_flag=vsf_perimeter(handle,flag);

int  new_flag;          new perimeter visibility flag
int  handle;            workstation handle
int  flag;              0 don't draw perimeter
                        1 show perimeter
```

DESCRIPTION

This function changes whether a border (or perimeter) is drawn the areas are filled with using the v_fillarea function and other functions that use the fill area attributes.

If flag is 1 then subsequent area fill calls will surround the area with a solid line in the current fill area colour. One function is an exception to this rule; the vr_recfl function *never* draws a border.

If flag is 0 then such borders are nct drawn.

RETURNS

This function returns the new value of the perimeter visibility flag.

SEE

vsf_color, vqf_attributes, v_fillarea

vsf_udpat — Set the user defined fill pattern

Class: VDI *Category: Fill Area Attributes*

SYNOPSIS

```
#include <vdi.h>

vsf_udpat(handle,pattern,planes);

int  handle;            workstation handle
short *pattern;         bit map to use
int  planes;            number of planes supplied
```

DESCRIPTION

This function changes the user defined fill pattern set using:

```
vsf_interior(handle,FIS_USER);
```

The planes parameter specifies the number of planes in this fill pattern. When using a monochrome device planes should be 1. Any planes that are not supplied will be zeroed when filling takes place.

The fill pattern is passed as 16 shorts for each plane, with the first short giving the top line of the pattern (the most significant bit being the leftmost pixel), and the last short giving the bottom line.

Note that only replace mode is valid when using a multi-plane fill pattern (see vswr_mode).

SEE

vsf_interior, vswr_mode

vsin_mode — Set input mode

Class: VDI *Category: Input Functions*

SYNOPSIS

```
#include <vdi.h>

new_mode=vsin_mode(handle,dev_type,mode);

int new_mode;       new mode selected
int handle;         workstation handle
int dev_type;       input device
int mode;           input mode
                    1 = request
                    2 = sample
```

DESCRIPTION

This function is used to set whether sample or request mode is to be used on a VDI input device. If you are using the AES at all for input, do not call these VDI functions as the AES will become confused.

The dev_type parameter should be one of

1	locator
2	valuator
3	choice
4	string

If the mode parameter is 1 then the device is set to request mode; if it is 2 it is set to sample mode.

RETURNS

This function returns the new mode set.

SEE

vrq_locator, vsm_locator, vrq_valuator, vsm_valuator, vrq_choice, vsm_choice, vrq_string, vsm_string

vsl_color

Set the line colour

Class: VDI *Category: Line Attributes*

SYNOPSIS

```
#include  <vdi.h>

new_col=vsl_color(handle,colour);

int  new_col;              new line colour set
int  handle;               workstation handle
int  colour;               new colour of line to use
```

DESCRIPTION

This function changes the colour of lines (as drawn with v_pline) and other routines that use the line attributes. The number of colours that can be selected depends on the screen resolution in use, and is returned by the v_opnvwk call. To change the colour palette use the vs_color function.

The line colours are shown in the table below. By default the control panel, if present, will change these to be the colours shown.

WHITE	White
BLACK	Black
RED	Red
GREEN	Green
BLUE	Blue
CYAN	Dark blue
YELLOW	Brown
MAGENTA	Dark green

LWHITE	Grey
LBLACK	Dark grey
LRED	Light blue
LGREEN	Blue green
LBLUE	Light purple
LCYAN	Dark purple
LYELLOW	Dark yellow
LMAGENTA	Light yellow

An L in a colour name indicates 'light'. LWHITE is really light grey and LBLACK is dark grey.

RETURNS

This function returns the line colour actually set. This will be 1 if you attempt to set a colour index that is too high for the current device.

SEE

vs_color

EXAMPLE

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

int  main(void)
{
  short  work_in[11]={1,1,1,1,1,1,1,1,1,1,2};
  short  work_out[57];
  short  handle;   /* virtual workstation handle */
  short  junk;
  short  pts[4]={10,20,30,40};

  appl_init();
  handle=graf_handle(&junk,&junk,&junk,&junk);
  v_opnvwk(work_in,&handle,work_out);
  if  (handle)
  {
    vsl_color(handle,RED); /*  red */
    v_pline(handle,2,pts); /*  draws a line between
                               (10,20) and (30,40) */
    evnt_keybd();
    v_clsvwk(handle);
  }
  return  appl_exit();
}
```

vsl_ends

Set the line end style

Class: VDI *Category: Line Attributes*

SYNOPSIS

```
    #include  <vdi.h>

    new_col=vsl_ends(handle,begin,end);

    int  handle;              workstation  handle
    int  begin;               starting  style
    int  end;                 ending  style
```

DESCRIPTION

This function changes how the beginning and ends of lines (as drawn with v_pline) and other graphics that use the line attributes. begin gives the style to use at the start of a line, whilst end gives the style for the end. The different styles are as follows:

SQUARE	▬▬▬▬
ARROWED	▬▬▬▶
ROUND	▬▬▬▬

SEE

vsl_type, vsl_color

EXAMPLE

```
    #include  <vdi.h>
    #include  <aes.h>
    int  main(void)
    {
      short  work_in[11]={1,1,1,1,1,1,1,1,1,1,2};
      short  work_out[57];
      short  junk,handle;  /* virtual  workstation  handle  */
      short  pts[4]={10,20,30,40};
      appl_init();
      handle=graf_handle(&junk,&junk,&junk,&junk);
      v_opnvwk(work_in,&handle,work_out);
      if  (handle)
      {
        vsl_ends(handle,ARROWED,ARROWED);
        v_pline(handle,2,pts);
        evnt_keybd();
        v_clsvwk(handle);
      }
      return  appl_exit();
    }
```

vsl_type, vsl_udsty

Set the line type

Class: VDI *Category: Line Attributes*

SYNOPSIS

```
    #include  <vdi.h>

    new_type=vsl_type(handle,type);
    vsl_udsty(handle,pattern);

    int  new_type;            type  of  line  set
    int  handle;              workstation  handle
    int  type;                new  line  type
    int  pattern;             used  defined  pattern
```

DESCRIPTION

These functions are used to change how lines (as drawn with v_pline) and other graphics that use the line attributes are drawn. The different line types are as follows:

SOLID	————————	Solid
LDASHED	——————	Long dash
DOTTED	—— ——	Dot
DASHDOT	———— ——	Dash dot
DASH	————	Dash
DASHDOTDOT	—— — —	Dash dash dot
USERLINE		User defined line as set by vsl_udsty.

The pattern parameter to vsl_udsty specifies the 16 bit user defined value to use. This is repeated along the line as for the standard patterns. SOLID is equivalent to a user-defined pattern of 0xFFFF. The user defined pattern is only used if USERLINE is set using vsl_type.

RETURNS

The vsl_type function returns the line type set.

SEE

vq_extnd

vsl_width

Set the line width

Class: VDI *Category: Line Attributes*

SYNOPSIS

```
#include <vdi.h>

new_size=vsl_width(handle,size);

int new_size;          width of line set
int handle;            workstation handle
int size;              new size of line to use
```

DESCRIPTION

This function changes the width of lines (as drawn with v_pline) and other graphics that use the line attributes. size, which gives the new line width, should be odd, otherwise the VDI will round the value down to the next odd value. Note that when using thickened lines the VDI may be unable to render line effects; vq_extnd can be used to determine whether this is possible.

RETURNS

This function returns the line width actually set.

SEE

vq_extnd

EXAMPLE

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

int main(void)
{
  short work_in[11]={1,1,1,1,1,1,1,1,1,1,2};
  short work_out[57];
  short junk,handle;  /* virtual workstation handle */
  short pts[4]={10,20,30,40};
  appl_init();
  handle=graf_handle(&junk,&junk,&junk,&junk);
  v_opnvwk(work_in,&handle,work_out);
  if (handle)
  {
    vsl_width(handle,3);   /* 3 pixels wide */
    v_pline(handle,2,pts); /* draws a line between
                              (10,20) and (30,40) */
    evnt_keybd();
    v_clsvwk(handle);
  }
  return appl_exit();
}
```

vsm_choice

Choice input in sample mode

Class: VDI *Category: Input Functions*

SYNOPSIS

```
#include <vdi.h>

status=vsm_choice(handle,xout);

int status;         choice status returned
int handle;         workstation handle
short *xout;        current value of choice
```

DESCRIPTION

This function is used to sample input from the 'choice' device. This is not implemented on the ST. Choice numbers vary from 1 to an implementation defined number. If you are using the AES at all for input, do not use the VDI input functions as the AES will become confused.

Before calling this function, you should call vsin_mode as follows:

```
vsin_mode(handle,3,2);
```

RETURNS

This function returns 1 if a choice input was made, otherwise returns 0.

SEE

vrq_choice, vsin_mode

vsm_color

Set the marker colour

Class: VDI *Category: Marker Attributes*

SYNOPSIS

```
#include  <vdi.h>

new_col=vsm_color(handle,colour);

int  new_col;               new marker colour set
int  handle;                workstation handle
int  colour;                new colour of marker to use
```

DESCRIPTION

This function changes the colour of markers as drawn with the v_pmarker function. The number of colours that can be selected depends on the screen resolution in use, and is returned by the v_opnvwk call. To change the colour palette use the vs_color function.

The colours are shown in the table below. By default the control panel, if present, will change these to be the colours shown below:

WHITE 0	White
BLACK f	Black
RED 1	Red
GREEN 2	Green
BLUE 4	Blue
CYAN	Dark blue
YELLOW 3	Brown
MAGENTA	Dark green

LWHITE	Grey
LBLACK	Dark grey
LRED	Light blue
LGREEN	Blue green
LBLUE	Light purple
LCYAN	Dark purple
LYELLOW	Dark yellow
LMAGENTA	Light yellow

An L in a colour name indicates 'light'. LWHITE is really light grey and LBLACK is dark grey.

RETURNS

This function returns the marker colour actually set. This will be 1 if you attempt to set a colour index that is too high for the current device.

SEE

vs_color, vsm_type, vsm_height, vqm_attributes, v_pmarker

EXAMPLE

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

int  main(void)
{
  short  work_in[11]={1,1,1,1,1,1,1,1,1,1,2};
  short  work_out[57];
  short  handle;  /* virtual workstation handle */
  short  junk;
  short  pts[4]={10,20,30,40};

  appl_init();
  handle=graf_handle(&junk,&junk,&junk,&junk);
  v_opnvwk(work_in,&handle,work_out);
  if  (handle)
  {
    vsm_type(handle,7);    /* diamond */
    vsm_height(handle,5);  /* height 5 */
    vsm_color(handle,RED);
    v_pmarker(handle,2,pts);  /* draws markers at
                                 (10,20) and (30,40) */
    evnt_keybd();
    v_clsvwk(handle);
  }
  return  appl_exit();
}
```

vsm_height — Set the marker height

Class: VDI *Category: Marker Attributes*

SYNOPSIS

```
#include  <vdi.h>

new_size=vsm_height(handle,size);

int  new_size;              height of markers set
int  handle;                workstation  handle
int  size;                  new size of marker to use
```

DESCRIPTION

This function changes the height (and thus width) of markers drawn with v_pmarker) to size pixels.

Note that the marker height has no effect on the 'dot' marker which is always exactly one pixel.

RETURNS

This function returns the marker height actually set.

SEE

vsm_color, vqm_attributes, vsm_type, v_pmarker

EXAMPLE

```
#include  <vdi.h>
#include  <aes.h>
int  main(void)
{
  short  work_in[11]={1,1,1,1,1,1,1,1,1,1,2};
  short  work_out[57];
  short  junk,handle;  /* virtual workstation handle */
  short  pts[4]={10,20,30,40};

  appl_init();
  handle=graf_handle(&junk,&junk,&junk,&junk);
  v_opnvwk(work_in,&handle,work_out);
  if  (handle)
  {
    vsm_type(handle,7);         /* diamond */
    vsm_height(handle,5);       /* height 5 */
    v_pmarker(handle,2,pts);    /* draws markers at
                                (10,20) and (30,40)  */
    evnt_keybd();
    v_clsvwk(handle);
  }
  return  appl_exit();
}
```

vsm_locator — Locator input in sample mode

Class: VDI *Category: Input Functions*

SYNOPSIS

```
#include  <vdi.h>

status=vsm_locator(handle,x,y,xout,yout,term);

int  status;            status  found
int  handle;            workstation  handle
int  x;                 initial  x  co-ordinate  of  locator
int  y;                 initial  y  co-ordinate  of  locator
short  *xout;           final  x  co-ordinate  of  locator
short  *yout;           final  y  co-ordinate  of  locator
short  *term;           terminator
```

DESCRIPTION

This function is used to sample input from the 'locator' device. On the ST this means mouse movement, keyboard and mouse button input. If you are using the AES at all for input, do not use the VDI input functions as the AES will become confused.

Before calling this function, you should call vsin_mode as follows:

```
vsin_mode(handle,1,2);
```

The x and y parameters give the position on screen where the mouse pointer will be displayed. If the user presses a key on the keyboard, term will contain the ASCII value of the key pressed. If the user clicks on a mouse button 32 will be returned in term for the left button and 33 for the right button. In any case the xout and yout parameters will contain the position of the mouse.

RETURNS

This function returns the following:

0	No change
1	Mouse has moved
2	Key (on keyboard or mouse) pressed
3	Both key press and movement.

Note that this function does not indicate whether a mouse button or a keyboard key was pressed.

SEE

vrq_locator, vsin_mode

EXAMPLE

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

int  main(void)
{
  short  work_in[11]={1,1,1,1,1,1,1,1,1,1,2};
  short  work_out[57];
  short  handle;   /* virtual workstation handle */
  short  junk;
  short  x,y;
  short  term;
  short  status;

  appl_init();
  handle=graf_handle(&junk,&junk,&junk,&junk);
  v_opnvwk(work_in,&handle,work_out);
  if  (handle>=0)
  {
    v_clrwk(handle);
    vsin_mode(handle,1,2);       /* locator,sample */
    x=50;  y=100;
    do
      status=vsm_locator(handle,x,y,&x,&y,&term);
    while  (status!=2);

    printf("Mouse position:  (%d,%d)  Key  pressed:%c\n"
      ,x,y,term);
    evnt_keybd();
    v_clsvwk(handle);
  }
  return  appl_exit();
}
```

vsm_string

String input in sample mode

Class: VDI *Category: Input Functions*

SYNOPSIS

```
#include  <vdi.h>

status=vsm_string(handle,max_len,echo,echo_xy,str);

int  status;          0=no  characters  available
                      n=characters  input
int  handle;          workstation  handle
int  max_len;         maximum  number  of  input  characters
int  echo;            0=  no  echo
                      1=  echo
short  *echo_xy;      co-ordinates  for  echoed  characters
char  *str;           string  input
```

DESCRIPTION

This function is used to sample input from the 'string' device. On the ST this means keyboard input. If you are using the AES at all for input, do not use the VDI input functions as the AES will become confused.

Before calling this function, you should call vsin_mode as follows:

```
vsin_mode(handle,4,2);
```

This function causes up to max_len characters to be input from the keyboard. The input will terminate if Return is pressed. The characters input are terminated by a 0 character. Thus str should be at least max_len+1 characters long.

If the echo parameter is not implemented on the ST. If it was implemented and a value of 1 was passed the characters typed would be echoed at position (echo_xy(0), echoxy(1)) on the device. It is however necessary to pass echo_xy as a 'real' pointer, otherwise bombs will result.

RETURNS

This function returns the number of characters input. This will be zero if there were none available.

SEE

vrq_string, vsin_mode

EXAMPLE

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

int  main(void)
{
  short  work_in[11]={1,1,1,1,1,1,1,1,1,1,2};
  short  work_out[57];
  short  handle;   /*  virtual  workstation  handle  */
  short  junk;
  short  pt[2]={100,100};

  char  str[7];

  appl_init();
  handle=graf_handle(&junk,&junk,&junk,&junk);
  v_opnvwk(work_in,&handle,work_out);
  if  (handle)
  {
    v_clrwk(handle);
    vsin_mode(handle,4,2); /*  string,sample  */

    while  (!vsm_string(handle,1,1,pt,str))
      ;

    printf("String  entered  was:  %s\n",str);
    evnt_keybd();
    v_clsvwk(handle);
  }
  return  appl_exit();
}
```

vsm_type

Set the marker type

Class: VDI *Category: Marker Attributes*

SYNOPSIS

```
#include  <vdi.h>

new_type=vsm_type(handle,type);

int  new_type;                type  of  marker  set
int  handle;                  workstation  handle
int  type;                    new  marker  type
```

DESCRIPTION

This function is used to change how markers (as drawn with v_pmarker) are drawn. The different marker types are as follows:

1	·	Dot
2	+	Plus
3	✳	Asterisk
4	□	Square
5	╳	Diagonal cross
6	◇	Diamond
7...		Device dependent

RETURNS

The function returns the marker type set.

SEE

vsm_color, vsm_height, v_pmarker

vsm_valuator

Valuator input in sample mode

Class: VDI *Category: Input Functions*

SYNOPSIS

```
#include <vdi.h>

vsm_valuator(handle,x,xout,term,status);

int handle;         workstation handle
int x;              initial value of valuator
short *xout;        final value of valuator
short *term;        terminator
short *status;      status found
```

DESCRIPTION

This function is used to sample input from the 'valuator' device. This is not implemented on the ST. Valuator numbers vary from 1 to 100. If you are using the AES at all for input, do not use the VDI input functions as the AES will become confused.

Before calling this function, you should call vsin_mode as follows:

```
vsin_mode(handle,2,2);
```

The status return values are as follows

0	Nothing happened
1	Valuator changed
2	Key press occurred

SEE

vrq_valuator, vsin_mode

vsp_message

Suppress palette messages

Class: VDI *Category: Palette Escape Functions*

SYNOPSIS

```
#include <vdi.h>

vsp_message(handle);

int handle;         workstation handle
```

DESCRIPTION

This function would suppress the screen messages produced by palette driver. However the palette escapes are not implemented on the ST.

vsp_save

Save palette driver state

Class: VDI *Category: Palette Escape Functions*

SYNOPSIS

```
    #include  <vdi.h>

    vsp_save(handle);

    int  handle;         workstation  handle
```

DESCRIPTION

This function would save the current state of the palette driver. However the palette escapes are not implemented on the ST.

vsp_state

Set palette driver state

Class: VDI *Category: Palette Escape Functions*

SYNOPSIS

```
    #include  <vdi.h>

    vsp_state(handle,port,num,lightness,interlace,
                                      planes,indices);

     int  handle;             workstation  handle
     int  port;               communication  ports
     int  num;                file  number
     int  lightness;          aperture  control  -3  to  +3
     int  interlace;          0=non-interlaced
                              1=interlaced
     int  planes;             number  of  planes
     short  *indices;         pointer  to  colour  indices
```

DESCRIPTION

This function would set the state of the palette driver. However the palette escapes are not implemented on the ST.

vst_alignment

Set the base line for graphics text

Class: VDI

Category: Text Attributes

SYNOPSIS

```
#include <vdi.h>

vst_aligment(handle,hin,vin,hout,vout);

int handle;           workstation handle
int hin;              horizontal alignment
int vin;              vertical alignment
short *hout;          horizontal alignment set
short *vout           vertical alignment set
```

DESCRIPTION

This function changes where co-ordinates passed to the v_justified and v_gtext functions refer to. The hin parameter specifies the horizontal alignment and should be one of:

0	Left justified (default).
1	Centre justified.
2	Right justified.

The vin parameter specifies the vertical alignment and should be one of:

0	Base line (default). The bottom of characters without descenders.
1	Half line. The top of lower case letters such as a and e.
2	Ascent line. The top of upper case letters such as A and E.
3	Bottom. The very bottom of the character cell.
4	Descent. The bottom of characters with descenders such as g and y.
5	Top. The very top of the character cell.

This function returns the values actually set in the hout and vout parameters.

SEE

v_gtext, v_justified

vst_color

Set the graphics text colour

Class: VDI

Category: Marker Attributes

SYNOPSIS

```
#include <vdi.h>

new_col=vst_color(handle,colour);

int new_col;          new text colour set
int handle;           workstation handle
int colour;           new colour of text to use
```

DESCRIPTION

This function changes the colour that text is drawn in using the v_justified and v_gtext functions. The number of colours that can be selected depends on the screen resolution in use, and is returned by the v_opnvwk call. To change the colour palette use the vs_color function.

The colours are shown in the table below. By default the control panel, if present, will change these to be the colours shown:

WHITE	White
BLACK	Black
RED	Red
GREEN	Green
BLUE	Blue
CYAN	Dark blue
YELLOW	Brown
MAGENTA	Dark green

LWHITE	Grey
LBLACK	Dark grey
LRED	Light blue
LGREEN	Blue green
LBLUE	Light purple
LCYAN	Dark purple
LYELLOW	Dark yellow
LMAGENTA	Light yellow

An L in a colour name indicates 'light'. LWHITE is really light grey and LBLACK is dark grey.

RETURNS

This function returns the text colour actually set. This will be 1 if you attempt to set a colour index that is too high for the current device.

SEE

vs_color, v_gtext, vqt_attributes, v_justified

EXAMPLE

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

int  main(void)
{
  short  work_in[11]={1,1,1,1,1,1,1,1,1,1,2};
  short  work_out[57];
  short  handle;   /*  virtual  workstation  handle  */
  short  junk;

  appl_init();
  handle=graf_handle(&junk,&junk,&junk,&junk);
  v_opnvwk(work_in,&handle,work_out);
  if  (handle)
  {
    vst_color(handle,RED);
    v_justified(handle,20,20,"Hello  World",100,1,1);
   /*  writes  hello  world  at  20,20  in  red*/
  .....
    v_clsvwk(handle);
  }
  return  appl_exit();
}
```

vst_effects

Set the graphics text effects

Class: VDI *Category: Text Attributes*

SYNOPSIS

```
#include  <vdi.h>

new_effects=vst_effects(handle,effects);

int  new_effects;          text  effects  set
int  handle;               workstation  handle
int  effects;              text  effects  to  use
```

DESCRIPTION

This function changes the appearance of the text that is drawn using the v_justified and v_gtext functions. The effects parameter is a bitmap, with mask components as follows:

Bit	Meaning
THICKENED	**Thicken**
SHADED	'Lighten'
SKEWED	*Skew*
UNDERLINED	Underline
OUTLINE	Outline
SHADOW	Shadowed

More than one effect may be set at once, but this can often look very ***unpleasant***!

RETURNS

This function returns the text effects set.

SEE

v_gtext, v_justified, linea8

vst_font

Select particular GDOS font

Class: VDI

Category: Text Attributes

SYNOPSIS

```
    #include  <vdi.h>

    set_font=vst_font(handle,font);

    int  set_font;        font  actually  set
    int  handle;          workstation  handle
    int  font;            font  index  requested
```

DESCRIPTION

This function should only be used when GDOS is loaded and changes the font that text is drawn in by the v_gtext and v_justified functions. You can find valid numbers for the font indices using the vqt_name function.

RETURNS

This function returns the font actually set.

SEE

vqt_name, vq_gdos

vst_height, vst_point

Set the text height

Class: VDI

Category: Text Attributes

SYNOPSIS

```
    #include  <vdi.h>

    vst_height(handle,h,charw,charh,cellw,cellh);
    set=vst_point(handle,p,charw,charh,cellw,cellh);

    int  set;              the  character  height  set
    int  handle;           workstation  handle
    int  h;                character  height  (pixels)
    int  p;                character  height  (points)
    short  *charh;         character  height  selected  (pixels)
    short  *charw;         character  width  selected  (pixels)
    short  *cellh;         cell  height  selected  (pixels)
    short  *cellw;         cell  width  selected  (pixels)
```

DESCRIPTION

This function changes the height (and thus width) of graphics text as drawn with v_gtext and v_justified).

The vst_height function is passed the height to select in pixels, whereas the vst_point function is passed the height in points (1/72th of an inch). If the function cannot use the given height then the next smallest is used. The character size selected is returned in charw and charh. cellw and cellh give the cell size in pixels.

Note that the vst_point function is preferred to vst_height as it uses a device portable measurement.

RETURNS

The function vst_point returns the height actually set in points.

SEE

vq_extnd

vst_load_fonts

Load GDOS fonts

Class: VDI *Category: Workstation Control*

SYNOPSIS

```
#include  <vdi.h>

add=vst_load_fonts(handle,select);

int  add;                 additional  fonts  loaded
int  handle;              workstation  handle
int  select;              reserved:  use  0
```

DESCRIPTION

This function is used to load GDOS fonts from disk; it is not required to load system fonts. The fonts are loaded into GEMDOS free memory, and thus you should check the value returned by this function to see how many fonts have been loaded. You can use this call more than once on the same workstation; the VDI will return 0 on subsequent calls.

The handle parameter should be the handle of the physical or virtual workstation, as returned by v_opnwk or v_opnvwk.

RETURNS

This function returns the number of additional fonts loaded.

SEE

v_opnvwk, v_opnwk, vq_gdos, vst_unload_fonts, vst_font, vqt_name

EXAMPLE

```
#include  <vdi.h>
int  main(void)
{
  short  work_in[11]={21,1,1,1,1,1,1,1,1,1,2};
  short  work_out[57];
  short  handle;
  int  fonts_loaded;

  if  (vq_gdos())
  {
    v_opnwk(work_in,&handle,work_out);
    if  (handle)
    {  /*  Now  load  printer  fonts*/
      fonts_loaded=vst_load_fonts(handle,0);
      .....
      v_clswk(handle);     /*  close  workstation  */
    }
  }
}
```

vst_rotation

Select rotated text

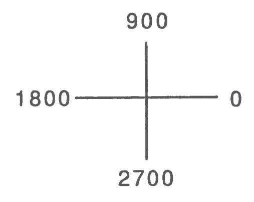
Class: VDI *Category: Text Attributes*

SYNOPSIS

```
#include  <vdi.h>

set_angle=vst_rotation(handle,angle);

int  set_angle;      rotation  angle  actually  set
int  handle;         workstation  handle
int  angle;          requested  angle  (0-3600)
```

DESCRIPTION

This function changes the angle at which graphics text is drawn by v_gtext and v_justified. Angles are specified in tenths of a degree, as follows:



If the device does support the angle requested, then the nearest possible value is selected and returned by the function.

The standard ST screen drivers only support values of 0, 900, 1800 and 2700. Do not pass a value greater than 3150, as a bus error may result.

RETURNS

This function returns the rotation angle actually set.

SEE

vq_extnd

vst_unload_fonts

Un-load GDOS fonts

Class: VDI *Category: Workstation Control*

SYNOPSIS

```
#include <vdi.h>

vst_unload_fonts(handle,select);

int handle;                 workstation handle
int select;                 reserved; use 0
```

DESCRIPTION

This function is used to free the space used by GDOS fonts that have been loaded from disk using vst_load_fonts.

The memory will only be freed when all the workstations using these fonts have either been closed or have called vst_unload_fonts. Thus there is no necessity to call this function, but it potentially gives an application more GEMDOS memory after the call.

The handle parameter should be the handle of the physical or virtual workstation, as returned by v_opnwk or v_opnvwk.

SEE

v_opnvwk, v_opnwk, vq_gdos, vst_load_fonts

EXAMPLE

```
#include <vdi.h>

int main(void)
{
  short  work_in[11]={21,1,1,1,1,1,1,1,1,1,2};
  short  work_out[57];
  short  handle;
  int  fonts_loaded;

  if (vq_gdos())
  {
    v_opnwk(work_in, &handle,work_out);
    if (handle)
    {
      /* Now load printer fonts*/
      fonts_loaded=vst_load_fonts(handle,0);
      vst_unload_fonts(handle,0);
      /* now we may have more free memory */
      v_clswk(handle);     /* close workstation */
    }
  }
}
```

vswr_mode

Set graphics drawing mode

Class: VDI *Category: Graphics Attributes*

SYNOPSIS

```
#include <vdi.h>

new_mode=vswr_mode(handle,mode);

int new_mode;             new writing mode
int handle;               workstation handle
int mode;                 mode to set
```

DESCRIPTION

This function is used to set the writing mode for all the graphics output functions. The possible values of mode are as follows:

MD_REPLACE 1	Replace mode ignores any existing data; the new data replaces the old pixel value.
MD_TRANS 2	Transparent mode only affects pixels where the pixel is already set.
MD_XOR 3	Exclusive OR mode changes the value of a pixel.
MD_ERASE 4	Reverse transparent mode only affects pixels where the source pixel is not set.

RETURNS

This function returns the new writing mode that has been set.

SEE

v_opnvwk, v_opnwk

4 GEMDOS Library

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

GEMDOS provides all the disk management, memory allocation and process management facilities traditionally available in an operating system. GEMDOS uses a consistent set of prefixes for its naming, these are:

Prefix	Function
C	Direct console, printer and auxiliary input/output.
D	Directory and disk management.
F	File management and manipulation.
M	Memory management.
P	Process creation and termination.
S	System inquiry and manipulation.
T	Time and date functions.

All functions in the GEMDOS 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.

Note that many of the functions listed in this section are known to have several bugs, where possible these have been documented as fully as possible under the 'Caveats' section.

In this section one function has been added to the standard GEMDOS selection, _mediach, which can be used to force the system to recognise a media change.

Many of the functions in GEMDOS have analogues in the main C library; using those functions can 'hide' many of the peculiarities and inconsistencies of GEMDOS. It will also make porting to Lattice C systems under other architectures simpler.

Cauxin $03 — Read a character from GEMDOS handle 2

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

SYNOPSIS

```
#include  <osbind.h>

x=Cauxin();

short x;  character obtained from standard aux
```

DESCRIPTION

The Cauxin function reads a character from GEMDOS handle 2 and returns it in the low byte of x. Note that the standard run time startup routine redirects this handle from the serial port (aux:) to the console device in order to provide a standard error facility.

SEE

Cconin, Cconrs, Crawio, Crawcin, Cnecin, Cconis, Bconin

CAVEATS

This function, when directed to aux:, can cause flow control on the RS232 port to break down and hence should be avoided. Also there is no way to indicate end-of-file when the handle has been redirected and the system may simply hang on reaching it.

Since this handle is used as the standard error handle by the standard C library, its use as a serial communication method is not recommended and the BIOS function Bconin should be used instead.

Cauxis $12 — Check status of GEMDOS handle 2

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

SYNOPSIS

```
#include  <osbind.h>

x=Cauxis();

short x;     status of standard auxiliary input
```

DESCRIPTION

This function checks the status of standard auxiliary input (GEMDOS handle 2) and returns the value -1 if at least one character is available. If no characters are available, Cauxis returns the value 0.

RETURNS

Cauxis returns -1 if at least one character is available, otherwise 0.

SEE

Cauxin, Bconin, Bconstat

CAVEATS

This handle is used as the standard error handle by the standard C library and hence its use as a serial communication method is not recommended and the BIOS function Bconstat should be used instead.

Cauxos $13 Check output status of GEMDOS handle 2

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

SYNOPSIS

```
#include <osbind.h>

x=Cauxos();

short x;      status of standard auxiliary output
```

DESCRIPTION

This function checks the status of the GEMDOS handle 2 and returns the value -1 if there is room for at least one character. If no characters may be sent, Cauxos returns the value 0.

RETURNS

The value -1 is returned if the stream attached to handle 2 is ready to receive a character, otherwise the value zero is returned.

SEE

Cauxout, Bconout, Bcostat

CAVEATS

This handle is used as the standard error handle by the standard C library and hence its use as a serial communication method is not recommended and the BIOS function Bcostat should be used instead.

Cauxout $04 Write a character to GEMDOS handle 2

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

SYNOPSIS

```
#include <osbind.h>

Cauxout(x);

short x;  character to be sent to standard aux
```

DESCRIPTION

The Cauxout function writes a character to GEMDOS handle 2. Note that the standard run time startup routine redirects this handle from the serial port (aux:) to the console device in order to provide a standard error facility.

SEE

Cconout, Cconin, Cconrs, Crawio, Cconis, Bconout

CAVEATS

This function, when directed to aux:, can cause flow control on the RS232 port to break down and hence should be avoided, also there is no way to check for characters successfully sent. Since this handle is used as the standard error handle by the standard C library, its use as a serial communication method is not recommended and the BIOS function Bconout should be used instead.

Cconin $01 Read a character from GEMDOS handle 0

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

SYNOPSIS

```
#include <osbind.h>

x=Cconin();

long x;     character obtained from standard in
```

DESCRIPTION

The Cconin function reads and echoes (to the standard *input*) a character from GEMDOS handle 0. Normally this will be attached to the keyboard, when the value returned in x gives the following information:

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

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. 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.

If the standard input stream has been redirected then only the low byte of x is valid and contains the character obtained from the stream *without* echoing.

This call checks for the special system keys (^C etc.) and so the process may be terminated as a result of this call.

RETURNS

As noted above.

SEE

Cconout, Cconis, Cconos, Cconrs, Cnecin, Crawio, Crawcin, Bconin

CAVEATS

There is no way to indicate end-of-file when the handle has been redirected and the system may simply hang on reaching it.

Cconis $0B Check status of standard input

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

SYNOPSIS

```
#include <osbind.h>

x=Cconis();

short x;     status of standard input
```

DESCRIPTION

This function checks the status of standard input (GEMDOS handle 0) and returns the value -1 if at least one character is available. If no characters are available, Cconis returns the value 0.

RETURNS

Cconis returns -1 if at least one character is available, otherwise 0.

SEE

Cconin, Bconin, Bconstat

Cconos $10 — Check status of standard output

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

SYNOPSIS

```
    #include  <osbind.h>

    x=Cconos();

    short  x;       status  of  standard  output
```

DESCRIPTION

This function checks the status of standard output (GEMDOS handle 1) and returns the value -1 if there is room for at least one character. If no characters may be sent, Cconos returns the value 0.

RETURNS

If the stream is directed to the console device (con:) then the call will always return -1. If however GEMDOS handle 1 has been redirected then this may not be the case and it may return 0 indicating that the output should cease.

SEE

Cconout, Bconout, Bcostat

Cconout $02 — Write a character to GEMDOS handle 1

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

SYNOPSIS

```
    #include  <osbind.h>

    Cconout(x);

    short  x;     character  to  write  to  standard  out
```

DESCRIPTION

The Cconout function writes the character x to the the stream attached to GEMDOS handle 1. Normally this will be attached to the screen, so that the character is printed on screen. Note that no line feed translation is performed on x and so to move to a new line both carriage return ('\r') and line feed ('\n') characters must be sent.

The high byte of x is reserved and must be zero for future compatibility.

This call checks for the special system keys (^C etc.) and so the process may be terminated as a result of this call.

SEE

Cconin, Crawio, Crawcin, Cconws, Bconout

CAVEATS

On version 1.0 and 1.2 of the operating system this call attempts to read a character from the standard *output* stream whilst attempting to process the special system keys. If handle 1 is directed to a write-only device (e.g. prn:) then the system will hang indefinitely.

Cconrs $0A Read a string from standard input

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

SYNOPSIS

```
    #include <osbind.h>

    Cconrs(buf);

    char *buf;    buffer to read characters into
```

DESCRIPTION

The Cconrs function reads a string from the standard input stream echoing it to the standard output stream. buf(0) contains the maximum number of characters that will be read.

On return buf(1) contains the number of characters actually read with the string starting at buf(2). Note that the string is not null terminated.

Cconrs always reads characters until the buffer is full or until it encounters a ^J or ^M (i.e. the Return key) which is discarded.

If the standard input stream is directed to the console this call reads an edited string from the console. The following key sequences are interpreted and acted upon:

^C	Cancel input line and terminate program
^H	Backspace and delete last character
DEL	Backspace and delete last character
^J	End input, do not place ^J in buffer
^M	End input, do not place ^M in buffer
^R	Echo input line and continue entry
^U	Echo input line and restart entry
^X	Cancel input line and restart entry

When the standard input stream has been re-directed to a file the call will return with buf(1) set to zero when end-of-file is reached.

RETURNS

The call returns with the number of characters obtained in buf(1) and a string starting at buf(2).

SEE

Cconout, Cconis, Cconos, Bconin, Bconout

CAVEATS

On version 1.0 and 1.2 of the operating system this call echoes the characters read from the standard input stream to the standard output even when it has been re-directed to a file.

Cconws $09 — Write string to standard output

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

SYNOPSIS

```
    #include  <osbind.h>

    Cconws(x);

    const  char  *str;          ASCIIZ  string  to  write  to
                                standard  out
```

DESCRIPTION

The Cconws function writes the ASCIIZ string str to the standard output stream calling Cconout for each character in the string, not including the terminating zero. Note that no line feed translation is performed on any of the characters and so to move to the start of a new line both carriage return ('\r') and line feed ('\n') characters must be sent.

This call checks for the special system keys (^C etc.) and so the process may be terminated as a result of this call.

SEE

Crawio, Cconout, Bconout

CAVEATS

On version 1.0 and 1.2 of the operating system this call attempts to read a character from the standard *output* stream whilst attempting to process the special system keys. If handle 1 is directed to a write-only device (e.g. prn:) then the system will hang indefinitely.

Cnecin $08 — Cooked input from standard in

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

SYNOPSIS

```
    #include  <osbind.h>

    x=Cnecin();

    long  x       character  obtained  from  standard  in
```

DESCRIPTION

The Cnecin function reads the first character from the standard input stream, without echoing it, however unlike Crawcin it *does* check for the special control keys. Normally this stream will be attached to the keyboard, when the value returned in x gives the following information:

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

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. Note that the shift key status is only returned if bit 3 in the system variable conterm is set. This defaults to off.

SEE

Crawio, Cconin, Crawcin, Cconrs, Cconis, Bconin

CAVEATS

There is no way to indicate end-of-file when the handle has been redirected and the system may simply hang on reaching it.

Cprnos $11 Check status of standard printer output

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

SYNOPSIS

```
#include <osbind.h>

x=Cprnos();

short x;      status of standard printer output
```

DESCRIPTION

This function checks the status of the standard printer output (GEMDOS handle 3) and returns the value -1 if there is room for at least one character. If no characters may be sent, Cprnos returns the value 0.

RETURNS

The value -1 is returned if the stream attached to handle 3 (normally prn:) is ready to receive a character, otherwise the value zero is returned.

SEE

Cconout, Bconout, Bcostat

Cprnout $05 Write a character to GEMDOS handle 3

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

SYNOPSIS

```
#include <osbind.h>

status = Cprnout(x);

short status;     status of printer
short x;          character to be sent to standard prn
```

DESCRIPTION

The Cprnout function writes a character to GEMDOS handle 3. Normally this will be attached to the printer, so that the character is printed. Note that no line feed translation is performed on this character and so to move to a new line it may be necessary to send both carriage return ('\r') and line feed ('\n') characters. Also note that no translation *whatsoever* is performed so that tab characters, for instance, are not expanded prior to sending to the device.

The high byte of x is reserved and must be zero for future compatibility.

RETURNS

The value 0 is returned if the call fails to write a character to the printer (e.g. not-ready), or non-zero if successful. Note that some older documentation incorrectly describes this function as 'returning void'.

SEE

Cconout, Bconout

Crawcin 807 — Raw input from standard in

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

SYNOPSIS

```
        #include  <osbind.h>

        x=Crawcin();

        long  x;      character  obtained  from  standard  in
```

DESCRIPTION

The Crawcin function reads the first character from the standard input stream, but unlike Cconin it never echoes the character and does not check for the special control keys. Normally this stream will be attached to the keyboard, when the value returned in x gives the following information:

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

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. Note that the shift key status is only returned if bit 3 in the system variable conterm is set. This defaults to off.

Note that when reading from the console via this handle the special system keys (^C etc.) are *not* checked.

The SEE

Crawio, Cconin, Cnecin, Cconrs, Cconis, Bconin

CAVEATS

There is no way to indicate end-of-file when the handle has been redirected and the system may simply hang on reaching it, also if you mix both Cconout and Crawcin calls, the system may become confused about the state of the special system keys.

Crawio 806 — Raw I/O to standard In/Out

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

SYNOPSIS

```
        #include  <osbind.h>

        y=Crawio(x);

        long  y;      character  obtained  when  x!=0x00ff
        short  x;     character  to  be  processed
```

DESCRIPTION

The Crawio function checks the value of x, if it is 0x00ff then a character is read from GEMDOS handle 0 (without echoing) if one is available. Normally this will be attached to the keyboard, when the value returned in y gives the following information:

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

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. Note that the shift key status is only returned if bit 3 in the system variable conterm is set. This defaults to off.

If no character is available then the value returned by Crawio is 0.

If x is not equal to 0x00ff, then the character is sent to GEMDOS handle 1, normally the screen device, when the return value y has no meaning. Note that when using this call the special system keys (^C etc.) are *not* checked so that it is, for example, impossible to pause the output using ^S.

The high byte of x is reserved and must be zero for future compatibility.

SEE

Cconout, Cconin, Cconrs, Cconis, Bconout, Bconin

CAVEATS

It is not possible to read zeroes, or write 0x00ffs via this function due to its definition. Also if you mix both Cconout and Crawio calls, the system may become confused about the state of the special system keys.

$39 $3A

Dcreate, Ddelete

Create/Delete GEMDOS folder

Class: GEMDOS *Category: Directory Functions*

SYNOPSIS

```
#include <osbind.h>

err = Dcreate(path); create new directory
err = Ddelete(path); delete old directory

long err;            error value
const char *path;    directory to create/delete
```

DESCRIPTION

The Dcreate function makes a new directory along the specified path. For example, if path is "c:\\abc\\def\\ghi", then a new directory named "ghi" is created in the path "c:\\abc\\def". The path may begin with a drive letter and a colon.

By contrast the Ddelete function removes an existing directory. Note that the directory *must* be empty otherwise the function will fail.

RETURNS

If the operation could not be performed a negative error code is returned, otherwise zero.

SEE

mkdir, rmdir

CAVEATS

Under 1.0 and 1.2 of TOS using Ddelete on a directory just created fails, a second Ddelete will successfully delete the directory. Also on these versions of TOS Dcreate does not always detect errors during directory construction and may partially build directories before failing.

Dfree $ 36

Get free disk space

Class: GEMDOS *Category: Disk Functions*

SYNOPSIS

```
#include <osbind.h>

error = Dfree(info,drive);

long error;             0 if successful
long *info;             disk information
short drive;            drive code
                        (0 => current drive)
```

DESCRIPTION

This function obtains allocation information from the specified disk drive. If a 0 is passed as drive, information is obtained about the current drive, otherwise drive should 1 for drive A, 2 for drive B, etc.

The pointer info should point to a buffer of 4 longwords, the DISKINFO structure in dos.h is suitable for this purpose and has the definition:

```
struct DISKINFO
{
  unsigned long free; /* number of free clusters */
  unsigned long cpd;  /* clusters per drive */
  unsigned long bps;  /* bytes per sector */
  unsigned long spc;  /* sectors per cluster */
};
```

RETURNS

A return value of 0 indicates success, otherwise a negative error code is returned.

CAVEATS

Under 1.0 and 1.2 of TOS this function is *very* slow on a hard disk and so should not be called routinely.

Dgetdrv, Dsetdrv $19 , $0E Get/Set default drive

Class: GEMDOS *Category: Disk Functions*

SYNOPSIS

```
    #include  <osbind.h>

    bmap  =  Dsetdrv(drive);       set  current  drive
    drive  =  Dgetdrv();           get  current  drive

    long  bmap;                    bitmap  of  mounted  drives
    short  drive;                  drive  number  to  get/set
```

DESCRIPTION

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

The Dgetdrv function returns the current drive code, using the same codes as Dsetdrv.

RETURNS

The function Dsetdrv returns a bitmap of mounted drives, bit 0 corresponds to drive A, bit 1 is drive B and so on. Note that although it returns a long GEMDOS currently only supports 16 devices, so the top 16 bits should be ignored.

The function Dgetdrv returns the code of the currently selected drive.

SEE

chgdsk, getdsk, Dgetpath, Dsetpath

Dgetpath, Dsetpath $47,$3B Get/Set current directory

Class: GEMDOS *Category: Directory Functions*

SYNOPSIS

```
    #include  <osbind.h>

    error  =  Dgetpath(buf,drive);
    error  =  Dsetpath(path);

    long  error;                   0  if  successful
    short  drive;                  drive  code
                                   (0  =>  current  drive)
    char  *buf;                    buffer  to  place  path  in
    const  char  *path;            path  to  change  to
```

DESCRIPTION

The Dgetpath function obtains the current path on the specified drive. Drive code 0 corresponds to the current drive, 1 to drive A, 2 is drive B and so on. The path is filled in in the buffer supplied in buf. Note that Dgetpath and Dgetdrv use different codes for the drives.

The Dsetpath function sets the current path to path. If the path string begins with a drive letter and a colon (:) then the directory for the specified drive is set.

RETURNS

A return value of 0 indicates success, otherwise a negative error code is returned.

SEE

chdir, getcd, getcwd

CAVEATS

Under *all* versions of TOS the Dsetpath function can become confused (causing logical drive assignments to be mixed up) if a drive letter and colon (:) are used in the path string, as such it is recommended that this feature be avoided.

Fattrib 843 Get/Set file attributes

Class: GEMDOS *Category: File Manipulation*

SYNOPSIS

```
#include <osbind.h>

fa = Fattrib(fname,flag,attr);

long fa;              file attributes
const char *fname;    name of file to manipulate
short flag;           get/set flag
                      0 => get attributes
                      1 => set attributes
short attr;           attributes when setting
```

DESCRIPTION

This function gets or sets the attribute byte for the specified file. The attributes (either returned in fa or set by attr) contain the following information:

Bit	Meaning
0	Read-only flag
1	Hidden file flag
2	System file flag
3	Volume label flag
4	Subdirectory flag
5	Archive flag (set if file has changed)
6	Reserved
7	Reserved

The archive flag is set whenever a file is created (or re-created) or when it has been written to using Fwrite (only on TOS 1.4 and above).

RETURNS

Fattrib returns the old attributes if successful or a negative error code if the operation could not be performed (e.g. the file does not exist).

SEE

chgfa, getfa

CAVEATS

The archive bit is only supported correctly in version 1.4 and above of the operating system.

Under 1.0 and 1.2 of TOS it is possible to use this function to perform illegal changes, e.g. removing the directory bit on a directory.

Fclose 83E — Close GEMDOS file

Class: GEMDOS *Category: File Manipulation*

SYNOPSIS

```
    #include <osbind.h>

    error = Fclose(handle);

    long error;             error status
    short handle;           file handle to close
```

DESCRIPTION

This function closes the file associated with the specified handle.

RETURNS

Fclose returns zero if the file was successfully closed, otherwise a negative error code.

SEE

Fcreate, Fopen, close

CAVEATS

Under 1.0 and 1.2 of TOS calling this function with an error value (e.g. as returned from Fopen) will usually result in a system crash. Also closing a standard handle (0-5) will leave the appropriate handle in an undefined state. On 1.4 and above the handle will revert to its default BIOS definition if closed.

Fcreate 83C — Create or truncate a file

Class: GEMDOS *Category: File Manipulation*

SYNOPSIS

```
    #include <osbind.h>

    handle = Fcreate(name,attr);

    long handle;              file handle
    const char *name;         name for file
    short attr;               attributes required
```

DESCRIPTION

This Fcreate function creates a new file (or truncates an old one) given by name. The attributes, attr, are made up of:

Bit	Meaning
0	Read-only flag
1	Hidden file flag
2	System file flag
3	Volume label flag
5	Archive flag (set if file has changed)

RETURNS

Fcreate returns a positive file handle if the file was successfully created, otherwise a *longword* negative error code. Note that word negative codes (0x0000ffff etc.) are used to signify devices such as con:.

SEE

Fopen, Fclose, creat

CAVEATS

Under TOS 1.0 creating a read-only file returns a read-only file handle. Also under 1.0 and 1.2 it is possible to create more than one volume name per root directory.

It may be useful under TOS 1.0 and 1.2 to set the archive bit as this is permitted on these versions. Under TOS 1.4 and above it is always set.

Fdatime 857 — Get/Set file time stamp

Class: GEMDOS — *Category: File Manipulation*

SYNOPSIS

```
#include  <osbind.h>

error  =  Fdatime(timeptr,fh,flag);

long  error;                error  value
short  *timeptr;            time/date  buffer
short  fh;                  handle  of  file
short  flag;                get/set  flag
                            0  =>  get  timestamp
                            1  =>  set  timestamp
```

DESCRIPTION

The Fdatime function gets or sets the timestamp of a file with handle fh. The timeptr buffer points to two words, the first of which gives the packed time, whilst the second holds the packed date. If flag is 0 the current timestamp is placed in the buffer, otherwise the timestamp is modified to that in the buffer.

The packed time longword may be represented by the bit fielded structure:

```
struct  timdat
{
  unsigned  hour:5;
  unsigned  minute:6;
  unsigned  second:5;
  unsigned  year:7;
  unsigned  month:4;
  unsigned  day:5;
}
```

Note that the time is stored in increments of two seconds and so the value obtained should be doubled to give a true number of seconds. Also note that the year is stored as an offset from 1980.

RETURNS

Fdatime returns zero if the file time was successfully interrogated/updated, otherwise a negative error code.

SEE

chgft, getft, ftunpk, ftpack

CAVEATS

Under 1.0 and 1.2 of TOS the return value of this function is not reliable and may indicate errors where none existed and as such it is probably best ignored.

Also beware that some older documentation incorrectly swaps the first two parameters to this call.

Fdelete $41

Delete file

Class: GEMDOS *Category: File Manipulation*

SYNOPSIS

```
    #include  <osbind.h>

    error  =  Fdelete(name);

    long  error;              error  value
    const  char  *name;       name  of  file  to  delete
```

DESCRIPTION

The Fdelete function deletes the named file. Note that only files may be deleted by this function, for directories you should use Ddelete.

RETURNS

The function returns zero if the file was successfully deleted, or a negative error number if the file could not be removed (e.g. was read-only).

SEE

Ddelete, remove, unlink

CAVEATS

If you attempt to delete a file that you have open, the file is closed and then deleted, however the handle *is not released* and hence will never be returned to GEMDOS. If you continue to use this handle there may be disastrous consequences.

Fdup $45

Duplicate standard file handle

Class: GEMDOS *Category: File Manipulation*

SYNOPSIS

```
    #include  <osbind.h>

    nh  =  Fdup(oh);

    long  nh;                 new  non-standard  handle
    short  oh;                standard  handle  (0-5)
```

DESCRIPTION

The Fdup function duplicates a standard file handle, (i.e. those numbered 0-5) and returns a non-standard handle (i.e. >6) which refers to the same device or file.

This function is most often used prior to calling the Fforce function so that the redirection may be 'undone'.

Note that when you have finished with this handle it should be released as normal via the Fclose function.

RETURNS

The function returns a new handle referring to the same device of file if successful, or a negative error number if an error occurred (e.g. no more handles left).

SEE

Fforce, dup

CAVEATS

Because this function always allocates a new handle, it is possible that when the process redirection depth becomes large the system may run out of handles, hence in general processes should consider communicating via intermediate files rather than redirected input and output.

Fforce $46

Redirect standard file handle

Class: GEMDOS *Category: File Manipulation*

SYNOPSIS

```
#include  <osbind.h>

error  =  Fforce(stdh,nstdh);

long  error;          error  value
short  stdh;          standard  handle  (0-5)
short  nstdh;         non-standard  handle  (>6  or  <0)
```

DESCRIPTION

The Fforce function forces the standard handle stdh to refer to the same file or device as the non-standard handle nstdh.

This function is generally used to force a child process to obtain its input from a file, or to send its output to a file.

RETURNS

The function returns a negative error number if an error occurred (e.g. invalid handle), or 0 if no error occurred.

SEE

Fdup, dup2

EXAMPLE

```
/*
 *  collect  a  command's  output  to  a  file
 */
#include  <osbind.h>
#include  <stddef.h>
long  collect(const  char  *command,const  char  *file)
{
  long  fh;
  long  ostdout,err;

  fh=Fcreate(file,0);
  if  (fh<0)
    return  fh;
  ostdout=Fdup(1);      /*  remember  current  stdout  */
  Fforce(1,fh);         /*  redirect  stdout  */
  err=Pexec(0,command,"",NULL);
  Fforce(1,ostdout);    /*  get  old  stdout  back  */
  Fclose(ostdout);      /*  release  handle  */
  /*  don't  close  fh  as  the  child  did  that  */
  return  err;
}
```

$2F $1A

Fgetdta, Fsetdta

Get/Set data transfer address (DTA)

Class: GEMDOS *Category: File Manipulation*

SYNOPSIS

```
#include  <osbind.h>

Fsetdta(dta);
dta  =  Fgetdta();

void  *dta;          pointer  to  DTA
```

DESCRIPTION

The Fsetdta function is used to change the data transfer address used by GEMDOS in the Fsfirst and Fsnext calls. By comparison the Fgetdta function returns the current data transfer address.

Note that the default DTA overlays some important system structures and the command line image in the base page, as such you should always move the DTA prior to using Fsfirst and Fsnext.

SEE

Fsfirst, Fsnext, chgdta, getdta

Fopen $3D

Open a GEMDOS file

Class: GEMDOS *Category: File Manipulation*

SYNOPSIS

```
    #include  <osbind.h>

    handle  =  Fopen(name,mode);

    long  handle;           file  handle
    const  char  *name;     name  of  file
    short  mode;            required  file  mode
```

DESCRIPTION

The Fopen function opens an existing file in the mode specified. The legal values for mode are:

0 (O_RDONLY)	Read-only access. No writes are allowed.
1 (O_WRONLY)	Write-only access. No reads are allowed.
2 (O_RDWR)	Read-write access. Both reads and writes are allowed.

Note that the names for the modes are the same as used by open. These values are found in the fcntl.h header file.

Note that in addition to files existing on a mounted drive, the special device names con:, aux: and prn: are recognised, giving access to the console, auxiliary and printer ports respectively.

RETURNS

Fopen returns a positive file handle if the file was successfully opened, otherwise a *longword* negative error code. Note that word negative codes (0x0000ffff etc.) are used to signify devices such as con:.

SEE

Fcreat, Fclose, open

Fread $3F

Read from an open file

Class: GEMDOS *Category: File Manipulation*

SYNOPSIS

```
    #include  <osbind.h>

    len  =  Fread(handle,count,buf);

    long  len;             length  read  from  file
    short  handle;         file  handle
    long  count;           length  to  read
    void  *buf;            buffer  to  read  into
```

DESCRIPTION

The Fread function reads from a file given by handle. count characters are read from the file into a buffer pointed to by buf. The process stops when either count characters have been read, or end of file has been reached.

If the handle specified points to a device (con: etc.) then the input is line buffered and Fread returns when a line has been read from the device.

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

Fread returns the number of characters successfully read, or a negative error code if a serious error occurred.

SEE

Fcreat, Fclose, open

CAVEATS

When reading from the keyboard you must provide some way to indicate end-of-file (e.g ^Z) also lines read from a device may be CR or LF terminated, but usually not CRLF terminated as is the TOS default.

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