v_pmarker — Draw one or more markers (polymarkers)

Class: VDI *Category: Graphics Output*

SYNOPSIS

```
#include <vdi.h>

v_pmarker(handle,n,pxyarray);

int handle;              workstation handle
int n;                   number of marker to plot
short *pxyarray          co-ordinate values
```

DESCRIPTION

This function is used to plot a series of markers at n points. The points are passed in pxyarray with (pxyarray(0), pxyarray(1)) giving the first point, (pxyarray(2), pxyarray(3)) giving the second point etc.

A single marker may be plotted using n=1.

How the markers are drawn depends on the marker attributes (see vsm_type etc.).

The handle parameter is the handle of the workstation to use, as usual.

SEE

vsm_type, vswr_mode, vsm_height, vsm_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, 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)
  {
    v_pmarker(handle,2,pts); /* draws dot markers
                  at (10,20) and (30,40) */
    v_clsvwk(handle);
  }
  return appl_exit();
}
```


v_rbox, v_rfbox — Output rounded rectangles

Class: VDI *Category: GDP Output*

SYNOPSIS

```
#include <vdi.h>

v_rbox(handle,pxyarray);
v_rfbox(handle,pxyarray);

int handle;              workstation handle
short *pxyarray;         co-ordinates of corners
```

DESCRIPTION

These 'Generalised Drawing Primitives' (GDPs) are used to draw rectangles with rounded corners whether filled or outlined.

The v_rbox function draws an outline of a rounded box using the line attributes (see vsl_color etc.) whereas the v_rfbox function draws a filled rounded rectangle using the fill area attributes (see vsf_color etc.). The corners of the box to draw are specified as (pxyarray(0), pxyarray(1)) and (pxyarray(2), pxyarray(3)). Unfortunately there is no way to set the size of the corners.

Devices don't necessarily support all GDPs. You can check that a particular GDP is available on a given device, by checking the values returned by v_opnwk or v_opnvwk. All GDPs are available in the standard ST screen modes.

The handle parameter is the handle of the workstation to use, as usual.

SEE

v_bar, vr_recfl, vsl_color, vsf_color

v_rvon, v_rvoff — Reverse video On/Off

Class: VDI

Category: Screen Escape Functions

SYNOPSIS

```
#include <vdi.h>

v_rvoff(handle);          turn off reverse video
v_rvon(handle);           turn on reverse video

int handle;               workstation handle
```

DESCRIPTION

v_rvon causes alpha text to appear in inverse video, i.e. with black and white reversed. It is equivalent to sending the ESC p VT52 code to the screen.

v_rvoff causes alpha text to appear in normal video, thus cancelling any call to v_rvon, and is equivalent to sending the ESC q VT52 code to the screen.

SEE

v_curtext

v_show_c — Display mouse cursor

Class: VDI

Category: Input Functions

SYNOPSIS

```
#include <vdi.h>

v_show_c(handle,reset);

int handle;               workstation handle
int reset;                0=reset count
                          1=use nested count
```

DESCRIPTION

This function can be used to display the mouse cursor. If your program is using the AES, you should use the graf_mouse call instead.

If the reset parameter to v_show_c is 0 then the mouse will be displayed regardless of the number of times that v_hide_c has been called previously. Otherwise v_show_c will only display the mouse form if it has been called at least as many times as v_hide_c.

The ability to reset this count is very tempting for lazy programmers; however if you use these VDI calls and the critical error handler is called then the mouse cursor will not appear; if you use graf_mouse then it will always appear.

SEE

graf_mouse, v_hide_c

v_updwk

Update workstation

Class: VDI

Category: Workstation Control

SYNOPSIS

```
#include <vdi.h>

v_updwk(handle);

int handle;                    workstation to update
```

DESCRIPTION

This function is not needed for screen devices. It is used for printers etc., to cause output to actually be printed. As such it is only useful when using GDOS.

After calling this function, you should normally call v_clrwk to skip to the next page.

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

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], handle;

  if (vq_gdos())
  {
    v_opnwk(work_in,&handle,work_out);
    if (handle)
    {
      /* Now write to the printer */
      .....
      v_updwk(handle);      /* output first page */
      v_clrwk(handle);      /* clear next one */
      ......
      v_updwk(handle);      /* output last page */
      v_clswk(handle);      /* close workstation */
    }
    else
      printf("Could not open printer");
  }
  else
    printf("Graphics on the printer needs GDOS");
  return 0;
}
```


v_write_meta

Write metafile item

Class: VDI

Category: Metafile Escape Functions

SYNOPSIS

```
#include <vdi.h>

v_write_meta(handle,intin_len,intin,ptsin_len,ptsin);

int handle;        metafile workstation handle
int intin_len;     length of intin array
short *intin;      intin array
int ptsin_len;     length of ptsin array
short *ptsin;      ptsin array
```

DESCRIPTION

This function writes an item to a metafile. To write standard items to a metafile you can use the standard calls with a metafile workstation handle.

v_write_meta can be used to write user defined opcodes which should have opcode numbers, passed in intin(0), greater than 100. This function is passed the standard GEMVDI intin and ptsin arrays.

The following sub-opcode numbers are pre-defined:

10	Start group.
11	End group.
49	Set no line style.
50	Set attribute shadow on.
51	Set attribute shadow off.
80	Start draw area type primitive.
81	End draw area type primitive.

SEE

vm_pagesize, vm_coords

vex_butv

Add mouse click routine

Class: VDI

Category: Vector Handling

SYNOPSIS

```
#include <vdi.h>

vex_butv(handle,but_addr,obut_addr);

int handle;                     workstation handle
int (*but_addr)(state);         new vector address
int (**obut_addr)(state);       old vector address
short state;                    mouse button state
```

DESCRIPTION

This function is used to add a routine that is called every time the mouse button status changes. This can be used to enable the AES to detect either right *or* left clicks.

This function is passed the routine to call in but_addr; vex_butv then supplies the application with the old routine.

The routine that is called should preserve all registers (although current versions of the operating system do not require any to be saved) and should call the old routine. It must not call the AES, VDI or GEMDOS and should avoid calling the BIOS and XBIOS as the operating system is not fully re-entrant. The routine is passed the current mouse button state (as described under vq_mouse) and should return the new state. This may be modified by the routine, as in the example below.

SEE

vq_mouse, evnt_button, evnt_multi

EXAMPLE

```
/*
 * enable right mouse button clicks to be detected
 */

#include <aes.h>
#include <vdi.h>
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>

int __regargs (*old)(short);

int handle;
volatile int real_state; /* contains true state */
```



```
__saveds __regargs int mouser(short state)
{
  __emit(0x48e7);       /* movem.l d0-d1/a0-a1,-(a7) */
  __emit(0xc0c0);

  if (state)
  {
    /* button pressed */
    real_state=state;
    if (state>1)
      state=1;          /* always return left */
  }
  state=old(state);
  __emit(0x4cdf);       /* movem.l (a7)+,d0-d1/a0-a1 */
  __emit(0x0303);
  return (int)state;
}


int main(void)
{
  short junk,kstate;

  appl_init();
  handle=graf_handle(&junk,&junk,&junk,&junk);
  vex_butv(handle,mouser,&old);
  do
  {
    evnt_button(1,1,1,&junk,&junk,&junk,&kstate);
    printf("%d ",real_state); /* display true state */
  }
  while (!kstate);
/* exit by holding down shift/alt/ctl and clicking */

  vex_butv(handle,old,&old);
  appl_exit();
  return 0;
}
```

vex_curv

Add mouse rendering routine

Class: VDI

Category: Vector Handling

SYNOPSIS

```
#include <vdi.h>

vex_curv(handle,cur_addr,ocur_addr);

int handle;                          workstation handle
int (*cur_addr)(x,y);                new vector address
int (**ocur_addr)(x,y);              old vector address
short x;                             mouse X position
short y;                             mouse Y position
```

DESCRIPTION

This function is used to add a routine that is called every time the mouse cursor is drawn. This could be used to draw your own cursor. The routine is passed the x and y positions for the cursor to draw.

The routine that is called should preserve all registers (although current versions of the operating system do not require any to be saved). It must not call the AES, VDI or GEMDOS and should avoid calling the BIOS and XBIOS as the operating system is not fully re-entrant. The routine is passed the position of the mouse cursor as its two parameters. If the routine does not draw its own mouse form then the original routine should be called.

SEE

vex_motv, graf_mouse

vex_motv

Add mouse movement routine

Class: VDI

Category: Vector Handling

SYNOPSIS

```
#include <vdi.h>

vex_motv(handle,mot_addr,omot_addr);

int handle;                          workstation handle
int (*mot_addr)(x,y);                new vector address
int (**omot_addr)(x,y);              old vector address
short x;                             mouse X position
short y;                             mouse Y position
```

DESCRIPTION

This function is used to add a routine that is called every time the mouse is moved. This can be used to produce a mouse accelerator like the one below.

This function is passed the routine to call in mot_addr; vex_motv then supplies the application with the old routine.

The routine that is called should preserve all registers (although current versions of the operating system do not require any to be saved) and should call the old routine. It must not call the AES, VDI or GEMDOS and should avoid calling the BIOS and XBIOS as the operating system is not fully re-entrant. The routine is passed the current mouse co-ordinates as its two parameters and should return the new x position in the register D0 and the new y position in D1. These may be modified by the routine, as in the example below.

SEE

vq_mouse, vex_butv

EXAMPLE

```
/*
 * increase the mouse speed by the 'speed' factor
 */

#include <vdi.h>
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>

int __regargs (*old)(short,short);

int handle;

short speed=2;
```

```
__saveds __regargs int mouser(short x,short y)
{
  static short prev_x=-1, prev_y=-1;
  long savea0,savea1;

  savea0=getreg(REG_A0);
  savea1=getreg(REG_A1);

  if (prev_x==-1)      /* initialise X position */
    prev_x=x;

  if (prev_y==-1)      /* initialise Y position */
    prev_y=y;

  x+=(x-prev_x)*speed;
  prev_x=x;

  y+=(y-prev_y)*speed;
  prev_y=y;

  old(x,y);

  putreg(REG_A1,savea1);
  putreg(REG_A0,savea0);

  putreg(REG_D1,y);
  return (int)x;
}

int main(void)
{
  short junk,kstate;

  appl_init();
  handle=graf_handle(&junk,&junk,&junk,&junk);
  vex_motv(handle,mouser,&old);
  do
    evnt_button(1,1,1,&junk,&junk,&junk,&kstate);
  while (!kstate);
/* exit by holding down shift/alt/ctl and clicking */

  vex_motv(handle,old,&old);
  appl_exit();
  return 0;
}
```


vex_timv

Add timer tick routine

Class: VDI

Category: Vector Handling

SYNOPSIS

```
#include <vdi.h>

vex_timv(handle,tim_addr,otim_addr,conv);

int handle;                    workstation handle
int (*tim_addr)(void);         new timer address
int (**otim_addr)(void);       old timer address
short *conv;                   milliseconds per tick
```

DESCRIPTION

This function is used to add a routine that is called every timer tick; currently this occurs at a rate of 50Hz (i.e. 50 times a second).

This function is passed the routine to call in tim_addr; vex_timv then supplies the application with the old routine and the number of milliseconds per clock tick in conv.

The routine that is called should preserve all registers (although current versions of the operating system do not require any to be saved) and should call the old routine. It must not call the AES, VDI or GEMDOS and should avoid calling the BIOS and XBIOS as the operating system is not fully re-entrant.

The example below uses the onbreak function to ensure that the timer vector is restored before the program terminates. This is essential as otherwise the timer will continue to run once your program is finished, with disastrous consequences.

SEE

onbreak

EXAMPLE

```
/*
 * implement a simple interrupt driven counter
 */

#include <aes.h>
#include <vdi.h>
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>

volatile int count;
int (*old)(void);
int handle;
```

```
__saveds int timer(void)
{
  __emit(0x48e7);      /* movem.l d0-d1/a0-a1,-(a7) */
  __emit(0xc0c0);

  count++;

  __emit(0x4cdf);      /* movem.l (a7)+,d0-d1/a0-a1 */
  __emit(0x0303);
  return old();
}

int do_end(void)
{
  short junk;

  vex_timv(handle,old,&old,&junk);
  appl_exit();
  return 0;
}

int main(void)
{
  short junk;

  appl_init();
  handle=graf_handle(&junk,&junk,&junk,&junk);
  vex_timv(handle,timer,&old,&junk);
  onbreak(do_end);

  /* exit via Ctrl-C */
  for (;;)
    printf("%d\n",count);
  return 0;
}
```


vm_coords

Change metafile co-ordinate system

Class: VDI

Category: Metafile Escape Functions

SYNOPSIS

```
#include <vdi.h>

vm_coords(handle,min_x,min_y,max_x,max_y);

int handle;      metafile workstation handle
int min_x;       x co-ordinate of top left corner
int min_y;       y co-ordinate of top left corner
int max_x;       x co-ordinate of bottom right corner
int max_y;       y co-ordinate of bottom right corner
```

DESCRIPTION

This function changes the co-ordinate system used by a metafile. The co-ordinates given to this function (min_x, min_y) to (max_x, max_y) are mapped to the page size width and height fields in the metafile header, as set by vm_pagesize.

Using this function allows arbitrary co-ordinate systems to be used (i.e. not simply NDC or RC). Naturally this function may only be used with metafiles.

SEE

vm_pagesize, v_opnwk

vm_filename

Change metafile name

Class: VDI

Category: Metafile Escape Functions

SYNOPSIS

```
#include  <vdi.h>

vm_filename(handle,fname);

int  handle;              metafile  workstation  handle
const  char  *fname;      filename  for  metafile
```

DESCRIPTION

This function changes the name of a given metafile handle. The default name is GEMFILE.GEM. This can only be used with metafile workstation handles under GDOS and is normally used immediately after the workstation is opened. Note that the old metafile, GEMFILE.GEM is not deleted by this call.

The new file name is passed in the string fname.

SEE

v_opnwk

vm_pagesize

Change metafile page size

Class: VDI

Category: Metafile Escape Functions

SYNOPSIS

```
#include  <vdi.h>

vm_pagesize(handle,width,height);

int  handle;              metafile  workstation  handle
int  width;               width  of  page
int  height;              height  of  page
```

DESCRIPTION

This function changes the width and height fields in the metafile header, and as such can only be used with metafile handles.

The width and height parameters give the size of the page in tenths of a millimetre.

SEE

v_opnwk

vq_cellarray

Inquire cell array definition

Class: VDI

Category: Inquire Functions

SYNOPSIS

```
#include <vdi.h>

vq_cellarray(handle,pxy,row_len,num_rows,el_used,
                        rows_used,status,colarray);

int handle;             workstation handle
short *pxy;             co-ordinates of area
int row_len;            length of rows in colarray
int num_rows;           number of rows in colarray
short* el_used;         elements used in colarray
short *rows_used;       rows used in colarray
short *status;          0 = no error
                        1 = error occurred
short *colarray         colour index array
```

DESCRIPTION

This function is not implemented on the ST. If it was, it would be used to produce a colour array from the given screen area.

SEE

v_cellarray

vq_chcells

Return alpha screen size

Class: VDI

Category: Screen Escape Functions

SYNOPSIS

```
#include <vdi.h>

vq_chcells(handle,row,columns);

int handle;         workstation handle
short *row;         number of alpha character rows
short *columns;     number of alpha character columns
```

DESCRIPTION

This function returns the number of rows and columns on the 'alpha', i.e. TOS-mode screen in the parameters row and column.

SEE

v_exit_cur

vq_color

Return current palette information

Class: VDI

Category: Inquire Functions

SYNOPSIS

```
#include <vdi.h>

vq_color(handle,col,flag,rgb);

int handle;         workstation handle
int col;            colour index
int flag            0=return colour requested
                    1=actual colour display on device
short *rgb;         values returned
```

DESCRIPTION

This function can be used to find the palette information for a given colour index, col, in RGB units.

If flag=0 then this function returns the RGB values that the user requested (via vs_color). If flag=1 then this function gives the RGB values as displayed in the device. The values returned are between 0 and 1000 and are as follows:

rgb(0)	Red
rgb(1)	Green
rgb(2)	Blue

If the colour index is out of range for this device then -1 is returned in rgb(0).

SEE

vs_color

vq_curaddress

Return alpha cursor position

Class: VDI

Category: Screen Escape Functions

SYNOPSIS

```
#include <vdi.h>

vq_curaddress(handle,row,column);

int handle;         workstation handle
short *row;         current cursor row
short *column;      current cursor column
```

DESCRIPTION

This function returns the current alpha cursor position in the parameters pointed to by row and column.

This facility of the escape functions has no equivalent VT52 code.

SEE

vs_curaddress

vq_extnd — Extended Inquire

Class: VDI

Category: Inquire Functions

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(19-56)	Reserved.

SEE

v_opnwk, v_opnvwk

vq_gdos

Determine whether GDOS is loaded

Class: Lattice

Category: Atari Escape Functions

SYNOPSIS

```
#include  <vdi.h>

res=vq_gdos();

int  res;          0 => GDOS is not loaded
                   !=0 => GDOS is loaded
```

DESCRIPTION

This function indicates 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.

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).

This function does not have an official name but uses an Atari approved method for determining the presence of GDOS.

SEE

v_opnwk

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;

  if  (vq_gdos())
  {
    /*
     * GDOS is present; try to open the printer
     */
    v_opnwk(work_in,  &handle,  work_out);
    ....
  }
  else
    printf("Graphics on the printer needs GDOS");
  return  0;
}
```


vq_key_s

Sample keyboard shift key status

Class: VDI

Category: Input Functions

SYNOPSIS

```
#include  <vdi.h>

vq_key_s(handle,status);

int  handle;                    workstation  handle
short  *status;                 shift  key  status
```

DESCRIPTION

This function can be used to find the current status of the shift, Ctrl and Alt keys. The current shift status is returned as a bit map in the status parameter.If a given bit is set it means that that button is down. The bits are as follows:

Bit	Meaning
0	Right shift key depressed.
1	Left shift key depressed.
2	Ctrl key depressed.
3	Alt key depressed.

SEE

evnt_button, Kbshift

vq_mouse

Sample mouse position and state

Class: VDI

Category: Input Functions

SYNOPSIS

```
#include <vdi.h>

vq_mouse(handle,status,x,y);

int handle;                 workstation handle
short *status;              button status
short *x;                   x co-ordinate of mouse
short *y;                   y co-ordinate of mouse
```

DESCRIPTION

This function can be used to find the current position of the mouse and whether the mouse buttons are up or down. The current mouse position is returned in (x, y).

The status parameter is a bit map giving which mouse buttons are depressed. If a given bit is set it means that that button is down. Bit 0 is the left mouse button, bit 1 is the right.

SEE

evnt_button

vq_scan

Return printer scan heights

Class: VDI

Category: Printer Escape Functions

SYNOPSIS

```
#include <vdi.h>

vq_scan(handle,grh,passes,alh,div);

int handle;          workstation handle
short *grh;          pixels per graphics scan
short *passes;       graphics head passes per page
short *alh;          pixels per alpha scan
short *div;          division factor for alh & grh
```

DESCRIPTION

This function obtains information about the printer given by handle. It is only available when passing a printer handle under GDOS.

The number of graphics passes required per page is returned in the parameter passes.

The number of pixels per graphics scan is given by grh/div and the number of passes per alpha scan is given by alh/div. Note that the division factor is returned so that devices may plot fractions of pixels on a pass.

SEE

v_opnwk, v_opnvwk

vq_tabstatus

Availability of tablet

Class: VDI

Category: Screen Escape Functions

SYNOPSIS

```
#include <vdi.h>

status=vq_tabstatus(handle);

int status;         0 = no tablet
                    1 = tablet available
int handle;         workstation handle
```

DESCRIPTION

This function returns whether a graphics tablet is available or not. On the ST this function returns 0 indicating that a tablet is not available.

RETURNS

This function returns 1 if a graphics tablet is available, 0 if not.

vqf_attributes

Return current fill area attributes

Class: VDI

Category: Inquire Functions

SYNOPSIS

```
#include <vdi.h>

vqf_attributes(handle,attr);

int handle;         workstation handle
short *attr;        values returned
```

DESCRIPTION

This function returns the current fill area attributes used by the v_fillarea call amongst others. The attr array should be large enough to accept 5 shorts (not 4 as sometimes specified in some old documentation). The attr array is filled in as follows:

attr(0)	Fill area interior style (see vsf_interior).
attr(1)	Fill area colour (see vsf_color).
attr(2)	Fill area style index (see vsf_style).
attr(3)	Writing mode (see vswr_mode).
attr(4)	Fill area perimeter status (see vsf_perimeter).

SEE

vsf_interior, vsf_color, vsf_style, vsf_perimeter, vswr_mode

vqin_mode

Return input mode for given device

Class: VDI

Category: Inquire Functions

SYNOPSIS

```
#include <vdi.h>

vqin_mode(handle,dev,mode);

int handle;      workstation handle
int dev;         device number
short *mode;     1 = request mode
                 2 = sample mode
```

DESCRIPTION

This function returns the current mode (input or sample) for the given VDI device. If you are using the AES at all for input, do not call the VDI input functions as the AES will become confused.

The dev parameter should be one of:

1	Locator
2	Valuator
3	Choice
4	String

SEE

vsin_mode

vql_attributes

Return current line attributes

Class: VDI

Category: Inquire Functions

SYNOPSIS

```
#include <vdi.h>

vql_attributes(handle,attr);

int handle;      workstation handle
short *attr;     values returned
```

DESCRIPTION

This function returns the current line attributes used by the v_pline call amongst others. The attr array should be large enough to accept 6 shorts (not 4 as sometimes specified in some old documentation. The attr array is filled in as follows:

attr(0)	Line type (see vsl_type).
attr(1)	Line colour (see vsl_color).
attr(2)	Writing mode (see vswr_mode).
attr(3)	End style for the start of lines (see vsl_ends).
attr(4)	End style for the end of lines (see vsl_ends).
attr(5)	Current line width (see vsl_width).

SEE

vsl_type, vsl_color, vswr_mode, vsl_ends, vsl_width, v_pline

vqm_attributes

Return current marker attributes

Class: VDI

Category: Inquire Functions

SYNOPSIS

```
#include <vdi.h>

vqm_attributes(handle,attr);

int handle;         workstation handle
short *attr;        values returned
```

DESCRIPTION

This function returns the current marker attributes used by the v_pmarker call amongst others. The attr array should be large enough to accept 5 shorts (not 4 as sometimes specified in some old documentation). The attr array is filled in as follows:

attr(0)	Marker type (see vsm_type).
attr(1)	Marker colour (see vsm_color).
attr(2)	Writing mode (see vswr_mode).
attr(3)	Current polymarker width (see vsm_height).
attr(4)	Current polymarker height (see vsm_height).

SEE

vsm_height, vsm_type, vsm_color, vswr_mode

vqp_films

Inquire palette film types

Class: VDI

Category: Palette Escape Functions

SYNOPSIS

```
#include <vdi.h>

vqp_films(handle,str);

int handle;         workstation handle
char *str;          names of film types
```

DESCRIPTION

This function would return a string containing the film types available. However, the palette escapes are not implemented on the ST.

vqp_state

Inquire palette driver state

Class: VDI

Category: Palette Escape Functions

SYNOPSIS

```
#include <vdi.h>

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

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

DESCRIPTION

This function would return information concerning the palette driver. However the palette escapes are not implemented on the ST.

vqt_attributes

Return current graphics text attributes

Class: VDI

Category: Inquire Functions

SYNOPSIS

```
#include <vdi.h>

vqt_attributes(handle,attr);

int handle;             workstation handle
short *attr;            values returned
```

DESCRIPTION

This function returns the current graphics attributes used by the v_gtext call amongst others. The attr array should be large enough to accept 10 shorts. The attr array is filled in as follows:

attr(0)	Current text face (see vst_font).
attr(1)	Text colour (see vst_color).
attr(2)	Text rotation (see vst_rotation).
attr(3)	Current horizontal alignment (see vst_alignment).
attr(4)	Current vertical alignment (see vst_alignment).
attr(5)	Writing mode (see vswr_mode).
attr(6)	Current character width (see vst_height, vst_point).
attr(7)	Current character height (see vst_height, vst_point).
attr(8)	Current cell width (see vst_height, vst_point).
attr(9)	Current cell height (see vst_height, vst_point).

SEE

vst_color, vst_height, vst_point, vst_font, vswr_mode, vst_alignment, vst_rotation

vqt_extent

Return the size of a piece of graphics text

Class: VDI

Category: Inquire Functions

SYNOPSIS

```
#include <vdi.h>

vqt_extent(handle,str,pts);

int handle;          workstation handle
const char*str       string whose size is to be found
short *pts;          values returned
```

DESCRIPTION

This function returns the screen area needed to display a string of graphics text using the current text attributes. This gives how much screen area will be used if v_gtext 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.

SEE

v_gtext, vqt_width

vqt_fontinfo

Return size information for the current font

Class: VDI

Category: Inquire Functions

SYNOPSIS

```
#include <vdi.h>

vqt_fontinfo(handle,min,max,dist,width,effects);

int handle;          workstation handle
short *min;          first character number in font
short *max;          last character number in font
short *dist;         distances
short *width;        maximum character width
short *effects;      effects
```

DESCRIPTION

This function returns information about the current font. The min and max parameters return the first and last characters in the font respectively. The width parameter gives the maximum cell width, not including any special effects. The dist parameter should point to an array of at least 5 shorts that will be filled in to give information on the distances between the base line and the following lines:

dist(0)	Very bottom of the cell descenders.
dist(1)	Bottom of characters with descenders.
dist(2)	The top of normal lower case letters.
dist(3)	The top of upper case letters.
dist(4)	The top of the cell.

The effects array should point to at least 3 shorts that will be filled in to give information on the effects, as set by vst_effects:

effects(0)	Additional x direction pixels for current text effects.
effects(1)	The number of pixels that the left hand of the character cell is slanted at the baseline.
effects(2)	The number of pixels that the top right is slanted relative to the base line.

SEE

vqt_extent

vqt_name

Return font name and index

Class: VDI

Category: Inquire Functions

SYNOPSIS

```
#include <vdi.h>

index=vqt_name(handle,num,name);

int index;      the font index
int handle;     workstation handle
int num;        font number
char *name;     font name
```

DESCRIPTION

This function requires GDOS for operation and 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_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 long which will be filled in to give the font name.

RETURNS

This function returns the font index.

SEE

vqt_extent

vqt_width

Return the width of an individual character

Class: VDI

Category: Inquire Functions

SYNOPSIS

```
#include <vdi.h>

status=vqt_width(handle,ch,cellw,left,right);

int status;     ch or -1 if error
int handle;     workstation handle
int ch;         character whose width is to be found
short *cellw;   cell width returned
short *left;    white space to the left
short *right;   white space to the right
```

DESCRIPTION

This function returns the width of a character together with the white space on either side of it.

The character is passed as the ch parameter and the width of its character cell is returned in cellw. The white space to the left of the character is returned in left and that to the right of the character is returned in right.

RETURNS

This function returns the character passed in ch or -1 if an error occurred.

SEE

vqt_extent

vr_recfl — Draw filled rectangle

Class: VDI *Category: Graphics Output*

SYNOPSIS

```
#include <vdi.h>

vr_recfl(handle,pxyarray);

int handle;              workstation handle
short *pxyarray;         co-ordinates of corners
```

DESCRIPTION

This function is used to fill a rectangle with corners (pxyarray(0), pxyarray(1)) and (pxyarray(2), pxyarray(3)) using the current fill area attributes (see vsf_interior etc.). However an outline (as set by vsf_perimeter) is *never* drawn with this function. To draw the same rectangle with an outline, use the v_bar function.

The handle parameter is the handle of the workstation to use, as usual.

SEE

v_fillarea, vsf_interior, vsf_style, vswr_mode, vsf_color, vsf_perimeter, vsf_udpat

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,junk;
  short rect[4]={10,20,100,100};

  appl_init();
  handle=graf_handle(&junk,&junk,&junk,&junk);
  v_opnvwk(work_in,&handle,work_out);
  if (handle)
  {
    v_clrwk(handle);  /* clear screen */
    vr_recfl(handle,rect);
    /* draws a rectangle with corners at (10,20),
       and (100,100) in black */
    evnt_keybd();     /* wait for a key */
  }
  return appl_exit();
}
```


vr_trnfm — Transform raster to/from standard format

Class: VDI *Category: Raster Functions*

SYNOPSIS

```
#include <vdi.h>

vr_trnfm(handle,src,dest);

int handle;      workstation handle
MFDB *src;       source memory form definition block
MFDB *dest;      destination memory form definition
                 block
```

DESCRIPTION

This function is used to transform an MFDB from standard to device specific form or vice versa. The structure of MFDBs is discussed under vro_cpyfm.

The mapping from colour indices to pixel values on sixteen colour devices such as the ST's low resolution screen are as follows:

0000	0	White
0001	2	Red
0010	3	Green
0011	6	Yellow
0100	4	Blue
0101	7	Magenta
0110	5	Cyan
0111	8	Light grey

1000	9	Dark grey
1001	10	Light red
1010	11	Light green
1011	14	Light yellow
1100	12	Light blue
1101	15	Light magenta
1110	13	Light cyan
1111	1	Black

The mapping from colour indices to pixel values on four colour devices such as the ST's medium resolution screen are as follows:

00	0	White
01	2	Red
10	3	Green
11	1	Black

The standard form consists of contiguous identically sized planes. The words within the planes have the most significant bit as the leftmost bit on the device. The planes start from the top and work down.

Note that this function may be used to perform in-place transformations, however it is *extremely* slow for large forms.

SEE

vrt_cpyfm

vro_cpyfm

Copy raster

Class: VDI

Category: Raster Functions

SYNOPSIS

```
#include <vdi.h>

vro_cpyfm(handle,wr_mode,pxyarray,src,dest);

int handle;          workstation handle
int wr_mode;         logic operation to perform
short *pxyarray;     co-ordinates of source and
                     destination rectangles
MFDB *src;           source memory form definition
                     block
MFDB *dest;          destination memory form definition
                     block
```

DESCRIPTION

This function is used to perform a 'blit' from one area of the screen to another, or to/from a user memory buffer.

The src and dest parameters indicate the source and destination forms to use. They are both pointers to Memory Form Definition Blocks or MFDBs. This structure is declared in vdi.h as follows:

```
typedef struct fdbstr
{
  void *fd_addr;          pointer to form
  short fd_w;             width of form
  short fd_h;             height of form
  short fd_wdwidth;       word width of form
  short fd_stand;         standard/device specific flag
  short fd_nplanes;       number of planes in form
  short fd_r1;            reserved
  short fd_r2;            reserved
  short fd_r3;            reserved
} MFDB;
```

The fd_addr field gives the address of the memory area to use or should be NULL if a physical device (such as the screen) is to be used.

The remaining parameters are only used when a memory area is being used; if you pass NULL in the fd_addr field then they will be filled in for you by the VDI.

The rest of the elements are as follows:

fd_w	Width of form in pixels.
fd_h	Height of form in pixels.
fd_wdwidth	Form width in words.
fd_stand	0 device specific format. 1 device independent format.
fd_nplanes	Number of bit planes.
fd_r1, fd_r2, fd_r3	Reserved.

The wr_mode parameter of vro_cpyfm function gives the logical operation to perform and should be one of:

Mode	Meaning
ALL_WHITE	0
S_AND_D	source AND destination
S_AND_NOTD	source AND (NOT destination)
S_ONLY	Replace source
NOTS_AND_D	(NOT source) AND destination
D_ONLY	destination
S_XOR_D	source XOR destination
S_OR_D	source OR destination
NOT_SORD	NOT (source OR destination)
NOT_SXORD	NOT (source XOR destination)
NOT_D	NOT destination
S_OR_NOTD	source OR (NOT destination)
NOT_S	NOT source
NOTS_OR_D	(NOT source) OR destination
NOT_SANDD	NOT (source AND destination)
ALL_BLACK	1

The pxyarray parameter is a pointer to 8 shorts with the following meanings:

pxyarray(0)	x co-ordinate of top left corner of source rectangle
pxyarray(1)	y co-ordinate of top left corner of source rectangle
pxyarray(2)	x co-ordinate of bottom right corner of source rectangle
pxyarray(3)	y co-ordinate of bottom right corner of source rectangle
pxyarray(4)	x co-ordinate of top left corner of destination rectangle
pxyarray(5)	y co-ordinate of top left corner of destination rectangle
pxyarray(6)	x co-ordinate of bottom right corner of destination rectangle
pxyarray(7)	y co-ordinate of bottom right corner of destination rectangle

The function then performs a blit from the first pxyarray rectangle located over the source MFDB to the second pxyarray in the destination MFDB.

SEE

vrt_cpyfm, linea7, lineae

vrq_choice

Choice input in request mode

Class: VDI

Category: Input Functions

SYNOPSIS

```
#include <vdi.h>

vrq_choice(handle,x,xout);

int handle;        workstation handle
int x;             initial value of choice
short *xout;       final value of choice
```

DESCRIPTION

This function is used to wait for 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,3,1);
```

SEE

vsm_choice, vsin_mode

vrq_locator

Locator input in request mode

Class: VDI

Category: Input Functions

SYNOPSIS

```
#include <vdi.h>

vrq_locator(handle,x,y,xout,yout,term);

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 wait for 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,1,1);
```

The x and y parameters give the position on screen where the mouse pointer will be displayed. The input terminates when the user either presses a key on the keyboard, in which case term will contain the ASCII value of the key pressed or a mouse button (in which case 32 for the left button and 33 for the right button) will be stored in term. In both cases the xout and yout parameters will contain the position of the mouse when the input terminated.

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

SEE

vsm_locator, vsin_mode

EXAMPLE

```
/*
 * watch mouse using vrq_locator
 */

#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;

  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,1,1);  /*  locator,request  */
    x=50;
    y=100;
    vrq_locator(handle,x,y,&x,&y,&term);
    vrq_locator(handle,x,y,&x,&y,&term);

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


vrq_string

String input in request mode

Class: VDI

Category: Input Functions

SYNOPSIS

```
#include  <vdi.h>

vrq_string(handle,max_len,echo,echo_xy,str);

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 wait for 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,4,1);
```

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 null character. Thus str should be at least max_len+1 characters long.

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), echo_xy(1)) on the device. It is however necessary to pass echo_xy as a 'real' pointer, otherwise bombs will result.

SEE

vsm_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>=0)
  {
    v_clrwk(handle);
    vsin_mode(handle,4,1);      /* string,request  */
    vrq_string(handle,5,1,pt,str);

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


vrq_valuator

Valuator input in request mode

Class: VDI

Category: Input Functions

SYNOPSIS

```
#include  <vdi.h>

vrq_valuator(handle,x,xout,term);

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

DESCRIPTION

This function is used to wait for 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,2,1);
```

SEE

vsm_valuator, vsin_mode

vrt_cpyfm

Copy raster from monochrome to colour

Class: VDI

Category: Raster Functions

SYNOPSIS

```
#include <vdi.h>

vrt_cpyfm(handle,wr_mode,pxyarray,src,dest,cols);

int handle;          workstation handle
int wr_mode;         logic operation to perform
short *pxyarray;     co-ordinates of source and
                     destination rectangles
MFDB *src;           source memory form definition
                     block
MFDB *dest;          destination memory form definition
                     block
short *cols;         colour indices for the 1s and 0s
                     in the data
```

DESCRIPTION

This function is used to 'blit' a monochrome image to a colour screen or device. This is similar to vro_cpyfm but the source MFDB must be that for a monochrome area; this function is not needed on monochrome devices.

The additional cols parameter points to two short values. cols(0) gives the colour index for the 1s in the source area and cols(1) gives that for the 0s.

SEE

vro_cpyfm

vs_clip

Set VDI clipping rectangle

Class: VDI

Category: Workstation Control

SYNOPSIS

```
#include <vdi.h>

vs_clip(handle,flag,pxyarray);

int handle;          workstation handle
int flag;            0 switch off clipping
                     1 enable clipping
short *pxyarray;     clipping rectangle
```

DESCRIPTION

This function is used to enable or disable 'clipping' by all the GEM VDI functions. When clipping is enabled (flag=1) the VDI will not draw outside the given rectangle pxyarray. pxyarray is set up as follows:

pxyarray(0)	x co-ordinate of one corner
pxyarray(1)	y co-ordinate of one corner
pxyarray(2)	x co-ordinate of diagonally opposite corner
pxyarray(3)	y co-ordinate of diagonally opposite corner

When disabling clipping (flag==0) pxyarray may be NULL.

Note that this function requires a VDI rectangle; the second corner is given *not* the width and height as for AES rectangles.

By default clipping is disabled when a workstation is opened.

SEE

v_opnvwk, v_opnwk

vs_color

Set the colour palette

Class: VDI

Category: Graphics Attributes

SYNOPSIS

```
#include <vdi.h>

new_mode=vs_color(handle,colour,rgb);

int handle;          workstation handle
int colour;          colour to change
short *rgb;          new rgb values (0-1000)
```

DESCRIPTION

This function is used to change the colour palette. The rgb parameter is normally an array of 3 values as follows:

rgb(0)	Red value (between 0-1000)
rgb(1)	Green value (between 0-1000)
rgb(2)	Blue value (between 0-1000)

The RGB values are passed as values between 0 and 1000 rather than those required by the ST hardware. The VDI will map these to the nearest actual value. The values set can be determined using vq_color.

SEE

vq_extnd, vq_color

EXAMPLE

```
#include <vdi.h>

/* assumes that handle is a valid VDI workstation */
...
...

short rgb[3]={0,0,1000};
vs_color(handle,0,rgb); /* set colour 0 to be blue */
....
```


vs_palette

Set IBM screen palette

Class: VDI

Category: IBM Escape Functions

SYNOPSIS

```
#include <vdi.h>

new=vs_palette(handle,pal);

int new;             new palette setting
int handle;          workstation handle
int pal;             0 = red, green, brown
                     1 = cyan, magenta, white
```

DESCRIPTION

This function is only used on IBM compatibles with CGA screens. It selects which palette to use, as above.

RETURNS

This function returns the palette selected.

SEE

vs_color

vsc_form

Redefine the mouse cursor

Class: VDI

Category: Input Functions

SYNOPSIS

```
#include  <vdi.h>

vsc_form(handle,newform);

int  handle;        workstation  handle
MFORM  *newform;    new  mouse  form
```

DESCRIPTION

This function is used to change the appearance of the mouse form on the screen. If you are using the AES, then you should use the AES graf_mouse function rather than this function.

The newform parameter is a pointer to a mouse form structure. This is defined, in vdi.h, as:

```
typedef  struct  mfstr
{
  short  mf_xhot;         x co-ordinate of hot spot
  short  mf_yhot;         y co-ordinate of hot spot
  short  mf_nplanes;      reserved should be 1
  short  mf_fg;           mask colour index normally 0
  short  mf_bg;           data colour index normally 1
  short  mf_mask[16];     bits of mask
  short  mf_data[16];     bits of data
}  MFORM;
```

mf_mask(0) gives the bit mask for the top line (16 bits) of the mouse form, mf_mask(1) that for the second line, etc.

Note that the mf_nplanes parameter gives the number of planes in the form and must always be 1 for the mouse cursor.

SEE

graf_mouse, lineab

vsf_color

Set the fill area colour

Class: VDI

Category: Fill Area Attributes

SYNOPSIS

```
#include  <vdi.h>

new_col=vsf_color(handle,colour);

int  new_col;           new fill area colour set
int  handle;            workstation handle
int  colour;            new fill area colour to use
```

DESCRIPTION

This function changes the colour that areas are filled with using the v_fillarea function and other functions that use the fill area 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 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.