# wind_close

Close a window

*Class: AES*

*Category: Window Handling*

## SYNOPSIS

```
#include <aes.h>

res=wind_close(handle);

int res;                error return;
int handle;             handle of window
```

## DESCRIPTION

This function closes a window with the given handle. This function must be passed a window handle returned by wind_create.

Once a window has been closed by this function, it will not be displayed on the screen; it may be re-opened using wind_open if desired. More usually it is followed by a call to wind_delete to delete the window.

## RETURNS

The function returns 0 if an error occurred or non-zero otherwise.

## SEE

wind_create, wind_open, wind_delete



# wind_create

Create a window

*Class: AES*

*Category: Window Handling*

## SYNOPSIS

```
#include <aes.h>

winhandle=wind_create(kind,x,y,w,h);

int winhandle;      handle of new window
int kind;           attributes of new window
int x;              x co-ordinate of full window
int y;              y co-ordinate of full window
int w;              width of full window
int h;              height of full window
```

## DESCRIPTION

This function creates a window and indicates the maximum size for the window.

The kind parameter gives the components that will be present in the window:

| | |
|---|---|
| NAME | Title bar with name. |
| CLOSE | Close box. |
| FULL | Full box. |
| MOVE | Can be moved. |
| INFO | Information line below title. |
| SIZE | Size box. |
| UPARROW | Up arrow. |
| DNARROW | Down arrow. |
| VSLIDE | Vertical slider. |
| LFARROW | Left arrow. |
| RTARROW | Right arrow. |
| HSLIDE | Horizontal slider. |

These are bit masks which should be 'ORed' together using | when more than one component is required.

This call does not actually display the window; to do so call the wind_open function. The x, y, w and h parameters are subsequently returned by the wind_get function with a WF_FXYWH parameter and so should normally be set up to be the entire usable area of the screen as returned by

```
wind_get(DESK,WF_CXYWH,&x,&y,&w,&h);
```

Once you have created a window with wind_create you should ensure that your program deletes the window using wind_delete before it terminates; otherwise your window will not be deleted until you return to the Desktop or a wind_new call is made.

## RETURNS

This function returns a window handle for use in identifying the window to other window handling routines, such as wind_open. If there are no more windows then a negative number will be returned. The maximum number of windows that may be open at one time is eight. This is a system wide limitation and thus your program should not try to open the full eight windows otherwise there will be none left for desk accessories.

Note that window handles are *not* the same as VDI workstation handles *or* GEMDOS handles.

## SEE

wind_open, wind_close, wind_delete, wind_get, wind_new



# wind_delete

Delete a window

*Class: AES* *Category: Window Handling*

## SYNOPSIS

```
#include <aes.h>

res=wind_delete(handle);

int res;                    error return;
int handle;                 handle of window
```

## DESCRIPTION

This function deletes a window with the given handle. This function must be passed a window handle returned by wind_create.

When a window is no longer required it should be closed using wind_close and then deleted using wind_delete.

## RETURNS

The function returns 0 if an error occurred or non-zero otherwise.

## SEE

wind_create, wind_open, wind_close

# wind_find

Find window 'under' given co-ordinate

*Class: AES*

*Category: Window Handling*

## SYNOPSIS

```
#include <aes.h>

handle=wind_find(x,y);

int handle;          found window handle
int x;               x co-ordinate to look for
int y;               y co-ordinate to look for
```

## DESCRIPTION

This function returns which window is 'under' the given x, y screen co-ordinates. The parameters are usually a mouse position that has been returned from another AES call.

## RETURNS

The function returns the window handle or 0 if the co-ordinates are over the desktop (i.e. the value DESK).

## SEE

evnt_button, evnt_multi



# wind_get

Find information about a window

*Class: AES*

*Category: Window Handling*

## SYNOPSIS

```
#include <aes.h>

res=wind_get(handle,request,x,y,w,h);

int res              error result
int handle;          window handle
int kind;            information to find
short *x;            depends on request
short *y;            depends on request
short *w;            depends on request
short *h;            depends on request
```

## DESCRIPTION

This function returns information about a window with the given handle depending on the value of the parameter request. Note that the standard binding expects *all* parameters to be passed, but as an extension to the standard a parameter of NULL may be used causing the relevant argument to be ignored.

In general the x, y, w and h parameters give the co-ordinates and size of a rectangle. Exceptions to this are noted in the table below:

| Name | Action |
|---|---|
| WF_WORKXYWH<br>WF_WXYWH | The current work area of the window is returned. |
| WF_CURRXYWH<br>WF_CXYWH | The current position and size of the window including borders. |
| WF_PREVXYWH<br>WF_PXYWH | The co-ordinates of the previous window size including borders. |
| WF_FULLXYWH<br>WF_FXYWH | The maximum size of the current window including borders. |
| WF_HSLIDE | x contains the current position of the horizontal slider between 1 and 1000. 1 is the left most position. |
| WF_VSLIDE | x contains the current position of the vertical slider between 1 and 1000. 1 is the top most position. |

| | |
|---|---|
| WF_TOP | x contains the handle of the top (active) window. |
| WF_FIRSTXYWH | The co-ordinates of the first rectangle in the window's rectangle list. Note that this function is called to find the first rectangle, subsequent rectangles are found via WF_NEXTXYWH. See the function rc_intersect for an example. |
| WF_NEXTXYWH | The co-ordinates of the next rectangle in the window's rectangle list. |
| WF_HSLSIZE | x contains the size of the horizontal slider relative to the horizontal scroll bar (1 to 1000). |
| WF_VSLSIZE | x contains the size of the vertical slider relative to the vertical scroll bar (1 to 1000). |
| WF_SCREEN | x and y give the address of the internal to the AES alert buffer and w and h give the length of this buffer. x and w are the 'high' words. Note that when using the 'blitter' (1.2) ROMs the length is zero and so this value should not be relied upon. |

## RETURNS

The function returns 0 if an error occurred or non-zero otherwise.

## SEE

wind_create, wind_set



# wind_info

Change the information line of a window

*Class: Lattice*

*Category: Window Handling*

## SYNOPSIS

```
#include <aes.h>

res=wind_info(handle, info);

int res;                    error return;
int handle;                 window handle
const char *info;           the new information line
```

## DESCRIPTION

This function is a special case of the wind_set call, which is easier to use than the standard binding but has the disadvantage of being non-portable to other C implementations.

This function is used to change the information line (beneath the title bar) of a window. The window to be modified is specified using the window handle returned by wind_create and the new string is given by the info parameter.

## RETURNS

The function returns 0 if an error occurred or non-zero otherwise.

## SEE

wind_set

## EXAMPLE

```
#include <aes.h>
....
int handle;
....
wind_info(handle,"New info line");
/* New info line will now appear on the info line */
```

# wind_new

Re-initialise window data structures

*Class: AES*

*Category: Window Handling*

## SYNOPSIS

```
#include <aes.h>

res=wind_new();

int res;                reserved
int handle;             handle of window
```

## DESCRIPTION

This function closes and deletes all windows, flushes all window buffers and returns to standard mouse usage including the wind_update count.

This is the function that is used by the Desktop to tidy up after an application quits and so should be used if your application needs to run a possibly badly behaved program. Unfortunately this call is only available on AES version 1.30 (Rainbow TOS) and above, so that it cannot be used by lazy programmers to return to a fixed state!

At the same time as calling this function you should also call wind_newdesk with a first parameter of NULL to reset the Desktop tree.

## RETURNS

The function return value is reserved.

## SEE

wind_newdesk, wind_set



# wind_newdesk

Use a new object tree for the Desktop

*Class: Lattice*

*Category: Window Handling*

## SYNOPSIS

```
#include <aes.h>

res=wind_newdesk(tree,object);

int res;                error return;
OBJECT *tree;           new object tree to draw
int object;             first object in tree to draw
```

## DESCRIPTION

This function is a special case of the wind_set call, which is easier to use than the standard binding but has the disadvantage of being non-portable to other C implementations.

This function is used to change the object tree (passed in the parameter tree) for the Desktop to draw. The first object drawn is object. The WTEST.C program provides an example of this.

Note that prior to termination you should reinstate the default tree by calling this function with the tree parameter set to NULL.

## RETURNS

The function returns 0 if an error occurred or non-zero otherwise.

## SEE

wind_set

## EXAMPLE

```
#include <aes.h>
....
OBJECT *tree;

wind_newdesk(tree,ROOT);
/* use our own tree */

wind_newdesk(NULL,ROOT);
/* use the Desktop's once more */
```

# wind_open

Open a window

*Class: AES*

*Category: Window Handling*

## SYNOPSIS

```
#include <aes.h>

res=wind_open(handle,x,y,w,h);

int res;          error return;
int handle;       handle of window
int x;            x co-ordinate of window initially
int y;            y co-ordinate of window initially
int w;            width of window initially
int h;            height of window initially
```

## DESCRIPTION

This function opens a window and displays it at its given initial size and position. These co-ordinates include the window's borders. This initial size need not necessarily be the maximum size as given by wind_create. This function must be passed a window handle returned by wind_create.

Note that wind_open does *not* display anything inside the window's work area, however it does cause a redraw event to be sent to the application hence you should wait until receiving this message before drawing the contents of your window.

## RETURNS

The function returns 0 if an error occurred or non-zero otherwise.

## SEE

wind_create, wind_close, wind_delete



# wind_redraw

Window redraw utility routine

*Class: Lattice*

*Category: Window Handling*

## SYNOPSIS

```
#include <aes.h>

res=wind_redraw(handle, rect, routine);

int res;                                  error return;
int handle;                               window handle
GRECT *rect;                              area to re-draw
int (*routine)(handle,p);                 routine to be called
GRECT *p;                                 sub-rectangle
```

## DESCRIPTION

This function can be used to simplify the handling of window redraw events. You need only supply a routine to draw a given rectangle within your window. wind_redraw will take care of the details such as the window's rectangle list, removing the mouse, and ensuring that the user can't pull down menus whilst the screen is being updated.

This routine requires the window's handle and a pointer to the rectangle returned by evnt_mesag or evnt_multi.

The routine that you supply takes a window handle as its first parameter and the rectangle, p, to be re-drawn as its second parameter. The function should normally return 1; if it returns 0 then your routine will not be called for any subsequent rectangles, so that you can use this if you need to abort re-drawing for any reason.

## RETURNS

The function returns 0 if an error occurred or non-zero otherwise.

## SEE

wind_get, evnt_mesag, evnt_multi, rc_intersect

# wind_set

Set window attributes

*Class: AES* *Category: Window Handling*

## SYNOPSIS

```
#include <aes.h>

res=wind_set(handle,request,x,y,w,h);

int handle;             window handle
int request;            parameter to set
short *x;               x co-ordinate of rectangle
short *y;               y co-ordinate of rectangle
short *w;               width of rectangle
short *h;               height of rectangle
```

## DESCRIPTION

This function sets a particular window attribute. Note that although the binding lists 4 (short *) parameters only as many as are required need be passed. The actions of the function are defined by the request parameter:

| Name | Action |
|---|---|
| WF_NAME | This sets the name or title of the window. Note that due to the 16 bit nature of the binding, the address character pointer passed must be split into it's high and low words. The ADDR macro is provided for this purpose. Alternatively the non-portable wind_title function may be used. |
| WF_INFO | This sets the information line of the window. Like WF_NAME the ADDR macro may be used to perform the word splitting required. Alternatively the non-portable wind_info function may be used. |
| WF_CURRXYWH<br>WF_CXYWH | Set the current position and size of the window including borders. All four parameters are required. Note that if as a result of this call the window size increases in either direction, or if a new part is uncovered then a redraw message will be sent to you by the AES. If you must always redraw as a result of this call then, rather than simply redrawing you should send yourself a redraw message which the AES will merge with any it may have generated automatically. |
| WF_HSLIDE | x contains the current position of the horizontal slider between 1 and 1000. 1 is the left most position. Note that you should take into account the length of the slider bar when adjusting this value. |
| WF_VSLIDE | x contains the current position of the vertical slider between 1 and 1000. 1 is the top most position. Note that you should take into account the length of the slider bar when adjusting this value. |
| WF_TOP | The window specified by handle is the window which you want the AES to place on top (i.e. make the active window). |
| WF_NEWDESK | This is used to change the object tree for the Desktop to draw. Like WF_NAME the ADDR macro may be used to perform the word splitting required. The first object to draw should be passed as the w parameter.<br><br>Alternatively the non-portable wind_newdesk function may be used. If you use this call, you should call it again prior to terminating with a (x, y) parameter of NULL to reinstate the default Desktop's tree. |
| WF_HSLSIZE | x contains the size of the horizontal slider (1 to 1000) or -1 for the default square box. |
| WF_VSLSIZE | x contains the size of the vertical slider (1 to 1000) or -1 for the default square box. |

## RETURNS

The function returns 0 if an error occurred or non-zero otherwise.

## SEE

wind_get, wind_title,wind_info,wind_newdesk

## EXAMPLE

```
#include <aes.h>

wind_set(handle,WF_NAME,ADDR("Window Title"));
/*
 * sets the window's title. Note that ADDR should
 * be used to ensure that the parameters are passed
 * on the stack correctly
 */

wind_set(handle,WF_INFO,ADDR("New info line");

wind_set(handle,WF_NEWDESK,ADDR(tree),ROOT);
/*
 * changes the Desktop tree to be the object tree
 * given by tree and draws the entire tree starting
 * at the root object. See WTEST.C for a complete
 * example.
 */
```



# wind_title

Change a window's title

*Class: Lattice*

*Category: Window Handling*

## SYNOPSIS

```
#include <aes.h>

res=wind_title(handle, title);

int res;                    error return
int handle;                 window handle
const char *title;          the new title
```

## DESCRIPTION

This function is a special case of the wind_set call, which is easier to use than the standard binding but has the disadvantage of being non-portable to other C implementations.

This function is used to change the window's title (or name). The window to be modified is specified using the window handle returned by wind_create and the new string is given by the title parameter.

## RETURNS

The function returns 0 if an error occurred or non-zero otherwise.

## SEE

wind_set

## EXAMPLE

```
#include <aes.h>
....
int handle;
....
wind_title(handle,"My window title");
/* My window title will now appear in the title bar*/
```

# wind_update

Window control utility

*Class: AES*

*Category: Window Handling*

## SYNOPSIS

```
#include <aes.h>

res=wind_update(request);

int res;                    error return
int request;                action to perform
```

## DESCRIPTION

This function is used to stop the user using menus, moving windows etc. whilst the application is outputting to the screen or when the application wants to do its own tracking of the mouse. These routines should be called strictly in pairs; note that they do nest, so that so long as the calls match there are no problems. If you call this function with a parameter END_MCTRL more times than BEG_MCTRL the machine may hang.

| | |
|---|---|
| BEG_UPDATE | Tells the operating system that the application is about to update the window and will wait until menus are not down before doing this. You should call this routine before writing to a window with the VDI. |
| END_UPDATE | Tells the operating system that the application has finished updating the window and that the user may pull down menus once more. Should be called after you have called the VDI if you called this routine with BEG_UPDATE. |
| BEG_MCTRL | Tells the operating system that the application is performing all mouse control itself and the AES will not let the user pull-down menus or click on windows. One use of this option is in desk accessories to prevent clicks 'falling through' to an application window below. |
| END_MCTRL | Tells the operating system that the application has finished doing its own mouse control and so the AES will let the user, once more, pull down menus and click on close boxes etc. This must always be called if you have called the routine with BEG_MCTRL beforehand. |



## RETURNS

The function returns 0 if an error occurred or non-zero otherwise.

## SEE

wind_create, wind_open, wind_close

# 3 VDI Library

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

The VDI provides the graphical primitives of the ST, dealing with, amongst other things, point plotting, line drawing, area filling and text drawing. It also has a user I/O system and deals with the mouse and keyboard. It is based on an older graphical kernel standard, GKS.

The VDI is named using to a consistent set of prefixes. All functions start with v and then optionally one or more characters:

| Prefix | Function type |
|---|---|
| v_ | Configuration, graphical output. |
| vex_ | Vector handling. |
| vm_ | Metafile specific routines. |
| vq_ | Workstation inquiry functions. |
| vqf_, vql_, vqm_, vqt_ | Graphical primitive attributes. |
| vqin_ | Inquire input mode. |
| vqp_ | Inquire palette attributes. |
| vr_, vro_, vrt_ | Raster operations. |
| vrq_ | Request mode input. |
| vs_ | Workstation configuration functions. |
| vsc_ | Configure mouse form. |
| vsf_ | Set fill area attributes. |
| vsin_ | Set input mode. |
| vsl_ | Set line attributes. |
| vsm_ | Set marker types, sample mode input. |
| vsp_ | Set palette attributes. |
| vst_ | Set text attributes. |
| vswr_ | Set writing mode. |

# v_alpha_text — Output text to printer

*Class: VDI* *Category: Printer Escape Functions*

## SYNOPSIS

```
#include <vdi.h>

v_alpha_text(handle,str);

int handle;          workstation handle
const char *str      string to output
```

## DESCRIPTION

This function outputs alpha text directly to a printer. It is only available when passing a printer handle under GDOS.

The string to be printed is passed in the parameter str and is passed directly to the printer apart from the 'escape' codes:

| | |
|---|---|
| "\f" | This causes a form feed as if by v_form_adv. |
| "\0220" | This two character sequence causes text to be output in bold. |
| "\0221" | This two character sequence cancels text emboldening. |
| "\0222" | This two character sequence causes text to be italicised. |
| "\0223" | This two character sequence cancels italic text. |
| "\0224" | This two character sequence causes text to be underlined. |
| "\0225" | This two character sequence cancels text underlining. |

Note that the octal sequence "\022" corresponds to the ASCII code 'DC2'.

## SEE

v_gtext, v_form_adv, vst_effects



# v_arc, v_pieslice — Output circular segment

*Class: VDI* *Category: GDP Output*

## SYNOPSIS

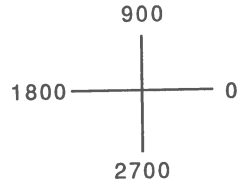
```
#include <vdi.h>

v_arc(handle,x,y,radius,begang,endang);
v_pieslice(handle,x,y,radius,begang,endang);

int handle;          workstation handle
int x;               x co-ordinate of centre
int y;               y co-ordinate of centre
int radius;          radius of circle
int begang;          start angle
int endang;          end angle
```

## DESCRIPTION

These 'Generalised Drawing Primitives' (GDPs) are used to draw a circular arc or a circular 'pie slice', starting at angle begang round to angle endang. Angles are specified in tenths of a degree as follows:




The v_arc function draws a circular arc using the line attributes (see vsl_color etc.) whereas the v_pieslice function draws a filled pie slice based on the fill area attributes (see vsf_color etc.).

The segment is drawn based on a circle with centre (x, y) and of the given radius in x-axis co-ordinates.

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_circle, 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 */

  appl_init();
  handle=graf_handle(&junk,&junk,&junk,&junk);
  v_opnvwk(work_in,&handle,work_out);
  if  (handle)
  {
    v_clrwk(handle); /* clear screen */
    v_arc(handle,100,100,30,0,900);
    /* draws a quarter of a circle */
    evnt_keybd();
    v_clsvwk(handle);
  }
  return  appl_exit();
}
```



# v_bar

Filled rectangle output

*Class: VDI*

*Category: GDP Output*

## SYNOPSIS

```
#include  <vdi.h>

v_bar(handle,pxyarray);

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

## DESCRIPTION

This 'Generalised Drawing Primitive' (GDP) 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.). This is exactly equivalent to an appropriate v_fillarea command.

Note that this function differs from vr_recfl in that the latter ignores any outline (as set by vsf_perimeter). The handle parameter is the handle of the workstation to use, as usual.

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.

## 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},  junk;
  short  rect[4]={10,20,100,100};work_out[57],  handle;
  appl_init();
  handle=graf_handle(&junk,&junk,&junk,&junk);
  v_opnvwk(work_in,&handle,work_out);
  if  (handle)  {
    v_clrwk(handle);          /* clear screen */
    v_bar(handle,rect);       /* draw rectangle */
    evnt_keybd();             /* wait for a key */
    v_clsvwk(handle);
  }
  return  appl_exit();
}
```

# v_bit_image

Write image file to printer

*Class: VDI*

*Category: Printer Escape Functions*

## SYNOPSIS

```
#include <vdi.h>

v_bit_image(handle,file,aspect,x_scale,y_scale,
                        h_align,v_align,pxyarray);

int handle;          workstation handle
const char *file;    image file to print
int aspect;          0 = ignore aspect ratio
                     1 = use file aspect ratio
int x_scale;         0 = fractional scaling on x-axis
                     1 = integer scaling on x-axis
int y_scale;         0 = fractional scaling on y-axis
                     1 = integer scaling on y-axis
int h_align;         Horizontal alignment
                          0 = left
                          1 = centre
                          2 = right
int v_align;         Vertical alignment
                          0 = top
                          1 = middle
                          2 = bottom
short *pxyarray;     rectangle giving area to print if
                     fractional scaling is used
```

## DESCRIPTION

This function prints a GEM .IMG file on a printer device. This can only be used with a printer handle under GDOS.

If the aspect flag is 1 then the aspect ratio from file will be used, thus giving the same aspect ratio as on the original device.

If fractional scaling is used then the VDI will output the image in the rectangle given by (pxyarray(0), pxyarray(1)) and (pxyarray(2), pxyarray(3)). If the image will not fit exactly than the h_align or v_align parameter will give the position within that rectangle.

## SEE

vq_scan, v_opnwk



# v_cellarray

Draw an array of cells

*Class: VDI*

*Category: Graphics Output*

## SYNOPSIS

```
#include <vdi.h>

v_cellarray(handle,pxyarray,rowlen,el_used,num_rows,
                                wrt_mode,colarray);

int handle;          workstation handle
short *pxyarray;     co-ordinate values
int rowlen;          length of rows in colarray
int el_used;         elements used in colarray
int num_rows;        number of rows in colour array
int wrt_mode;        writing operation to perform
short *colarray      colour array values
```

## DESCRIPTION

This function is not actually implemented on the ST. It would be used to plot an array of different coloured cells, placed in a rectangle with top left corner (pxyarray(0), pxyarray(1)) and bottom right corner (pxyarray(2), pxyarray(3)).

Normally colarray would be defined to be:

```
short colarray[num_rows*el_used];
```

The writing mode is as specified for the vswr_mode function.

## SEE

vswr_mode

# v_circle

Draw a circle

*Class: VDI*

*Category: GDP Output*

## SYNOPSIS

```
#include <vdi.h>

v_circle(handle,x,y,radius);

int handle;             workstation handle
int x;                  x co-ordinates of centre
int y;                  y co-ordinate of centre
int radius;             radius of circle
```

## DESCRIPTION

This 'Generalised Drawing Primitive' (GDP) is used to draw a circle using the fill area attributes (vsf_color etc.). The circle is drawn with centre (x, y) and of the given radius in x-axis co-ordinates.

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_ellipse, vsf_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],junk,handle;
  appl_init();
  handle=graf_handle(&junk,&junk,&junk,&junk);
  v_opnvwk(work_in,&handle,work_out);
  if (handle)
  {
    v_clrwk(handle); /* clear screen */
    v_circle(handle,100,100,30);
    /* draws a filled circle centred at (100,100),
       radius 30 pixels in black */
    evnt_keybd();
    v_clsvwk(handle);
  }
  return appl_exit();
}
```



# v_clear_disp_list

Clear display list

*Class: VDI*

*Category: Printer Escape Functions*

## SYNOPSIS

```
#include <vdi.h>

v_clear_disp_list(handle);

int handle;         workstation handle
```

## DESCRIPTION

This function clears the printer display list and can only be used with GDOS. Printer output under GDOS works by storing a list of items to be printed and then building up a bit map when a page is printed.

This function is similar to calling v_clrwk except that a form feed is not sent to the printer.

## SEE

v_updwk, v_clrwk

# v_clrwk

Clear workstation

*Class: VDI*

*Category: Workstation Control*

## SYNOPSIS

```
#include <vdi.h>

v_clrwk(handle);

int handle;                workstation to clear
```

## DESCRIPTION

This function is used to clear a physical workstation that has been opened using v_opnvwk or v_opnwk. The whole of the screen (or page on the printer) will be set to colour 0.

There is no need to call this function after opening a physical workstation, as the VDI will do this for you. However, virtual workstations are not cleared when they are opened, nor should you clear them in general as this call operates on the whole workstation and not just your virtual workstation.

## SEE

v_opnvwk, v_opnwk

## 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)
  {
    v_clrwk(handle); /* clear the screen */
    .....
    v_clsvwk(handle);
  }
  return appl_exit();
}
```



# v_clsvwk

Close virtual workstation

*Class: VDI*

*Category: Workstation Control*

## SYNOPSIS

```
#include <vdi.h>

v_clsvwk(handle);

int handle;                workstation handle to close
```

## DESCRIPTION

This function is used to close a virtual workstation that has been opened using v_opnvwk. This should always be called if v_opnvwk has been used.

## SEE

v_opnvwk, v_opnwk, v_clswk

## 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)
  {
    /* Now the main program */
    .....
    v_clsvwk(handle);
  }
  return appl_exit();
}
```

# v_clswk

Close physical workstation

*Class: VDI*

*Category: Workstation Control*

## SYNOPSIS

```
#include  <vdi.h>

v_clswk(handle);

int  handle;                workstation  handle  to  close
```

## DESCRIPTION

This function is used to close a physical workstation that has been opened using v_opnwk. This should always be called if v_opnwk has been used, but after any virtual workstations have been closed using v_clsvwk.

## SEE

v_opnwk, v_clswvk, vq_gdos

## EXAMPLE

```
#include  <vdi.h>
#include  <stdio.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);
    if  (handle)
    {
      /* Now write to the printer */
      .....
      v_clswk(handle);
    }
    else
      printf("Could  not  open  printer");
  }
  else
    printf("Graphics  on  the  printer  needs  GDOS");
  return  0;
}
```



# v_contourfill

'Seed' fill an area

*Class: VDI*

*Category: Graphics Output*

## SYNOPSIS

```
#include  <vdi.h>

v_contourfill(handle,x,y,colour);

int  handle;              workstation  handle
int  x;                   x  co-ordinate  of  start  point
int  y;                   y  co-ordinate  of  start  point
int  colour;              colour  to  search  for
```

## DESCRIPTION

This function is used to 'seed' fill an area of the screen starting at (x, y). Normally the fill continues until a pixel of the given colour (or the edge of the screen or paper) is found. Thus the colour is used as the border of the area to be filled.

If colour is negative then the fill continues until pixels other than the original colour in (x, y) is found. Thus this can be used to replace an area of one colour with another colour. How the area is drawn depends on the fill area attributes (see vsf_interior etc.).

## SEE

vsf_interior, vsf_style, vswr_mode, vsf_color, vsf_perimeter, vsf_udpat

## EXAMPLE

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

int  main(void)
{
  ....
  short  pts[6]={10,20,100,40,20,100};
  ...
  {
    v_clrwk(handle);  /* clear screen */
    v_fillarea(handle,3,pts);
    /* draws a triangle with corners at (10,20),
      (100,40) and (20,100) in black */
    vsf_color(handle,GREEN);
    v_contourfill(handle,30,50,WHITE);  /*  fill  with
                                            GREEN */
    evnt_keybd(); /* wait for a key */
    v_clsvwk(handle);
  }
  return  appl_exit();
}
```

# v_curhome, vs_curaddress — Position cursor

*Class: VDI* *Category: Screen Escape Functions*

## SYNOPSIS

```
#include <vdi.h>

v_curhome(handle);
vs_curaddress(handle,row,column);

int handle;        workstation handle
int row;           new row for cursor
int column;        new column for cursor
```

## DESCRIPTION

These functions are used to position the the alpha cursor on the screen. v_curhome causes the alpha cursor to move to the top left corner of the screen. This is equivalent to sending the ESC H VT52 code to the screen.

vs_curaddress causes the alpha cursor to move to the given row and column (both starting at 1). This is equivalent to sending the ESC Y VT52 code and the appropriate co-ordinates to the screen.

## SEE

v_curleft, v_curright, v_curup, v_curdown



# v_curleft, v_curright — Alpha cursor Left/Right

*Class: VDI* *Category: Screen Escape Functions*

## SYNOPSIS

```
#include <vdi.h>

v_curleft(handle);
v_curright(handle);

int handle;        workstation handle
```

## DESCRIPTION

v_curleft causes the alpha cursor to move left one character, or to remain at the first cursor position if it is already there. This is equivalent to sending the ESC D VT52 code to the screen.

Alternatively to move the cursor right one character, or to remain at the last cursor position if it is already there, the v_curright function is used, which is identical to the ESC C VT52 code.

## SEE

v_curdown, v_curup

# v_curtext — Output cursor addressable text

*Class: VDI* *Category: Screen Escape Functions*

## SYNOPSIS

```
#include <vdi.h>

v_curtext(handle,str);

int handle;         workstation handle
const char *str;    string to output
```

## DESCRIPTION

This function causes the 'alpha' text given by str to be written at the current alpha cursor position. The text will be displayed in reverse video if the v_rvon function has been called.

## SEE

vs_curaddress, v_rvon, v_rvoff



# v_curup, v_curdown — Alpha cursor Up/Down

*Class: VDI* *Category: Screen Escape Functions*

## SYNOPSIS

```
#include <vdi.h>

v_curdown(handle);      move alpha cursor down
v_curup(handle);        move alpha cursor up

int handle;             workstation handle
```

## DESCRIPTION

v_curdown causes the alpha cursor to move down one line, or to remain on the bottom line if it is already there. This is equivalent to sending the ESC B VT52 code to the screen.

Alternatively to move the cursor up one line, or to remain on the top line if it is already there, the v_curup function is used, which is identical to the ESC A VT52 code.

## SEE

v_curleft, v_curright

# v_dspcur, v_rmcur — Show/Hide mouse cursor

*Class: VDI* *Category: Screen Escape Functions*

## SYNOPSIS

```
#include <vdi.h>

v_dspcur(handle,x,y);
v_rmcur(handle);

int handle;        workstation handle
int x;             x co-ordinate of cursor
int y;             y co-ordinate of cursor
```

## DESCRIPTION

The v_dspcur function displays the mouse cursor on the screen at the position (x, y). By contrast v_rmcur removes the last mouse cursor displayed.

Normally these functions are not called but the AES graf_mouse routine used instead. If you are using the VDI to control the mouse then use the v_show_c and v_hide_c calls.

## SEE

v_hide_c, v_show_c, graf_mouse



# v_eeol, v_eeos — Erase to end of alpha line/screen

*Class: VDI* *Category: Screen Escape Functions*

## SYNOPSIS

```
#include <vdi.h>

v_eeol(handle);

int handle;        workstation handle
```

## DESCRIPTION

The v_eeol function causes the screen line to be cleared from the current cursor position. It does not change the current cursor position. This is equivalent to sending the ESC K VT52 code to the screen.

By contrast the v_eeos function causes the screen to be cleared from the current cursor position. It does not change the current cursor position. It is equivalent to sending the ESC J VT52 code to the screen.

## SEE

vs_curaddress, vq_curaddress

# v_ellarc, v_ellpie — Output elliptical segment

*Class: VDI* *Category: GDP Output*

## SYNOPSIS

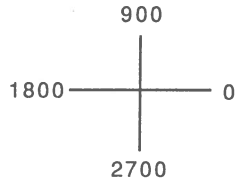
```
#include <vdi.h>

v_ellarc(handle,x,y,xradius,yradius,begang,endang);
v_ellpie(handle,x,y,xradius,yradius,begang,endang);

int handle;          workstation handle
int x;               x co-ordinates of centre
int y;               y co-ordinate of centre
int xradius;         x radius of ellipse
int yradius;         y radius of ellipse
int begang;          start angle
int endang;          end angle
```

## DESCRIPTION

These 'Generalised Drawing Primitives' (GDPs) are used to draw an elliptical arc or an elliptical 'pie slice', starting at angle begang to angle endang. Angles are specified in tenths of a degree as follows:




The v_ellarc function draws an elliptical arc using the line attributes (see vsl_color etc.) whereas the v_ellpie function draws a filled elliptical pie slice based on the fill area attributes (see vsf_color etc.). To draw circular arcs and pie slices use v_arc and v_pieslice.

The segment is drawn based on an ellipse with centre (x, y) and of the given xradius and yradius.

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_circle, v_arc, v_pieslice, vsl_color, vsf_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 */

  appl_init();
  handle=graf_handle(&junk,&junk,&junk,&junk);
  v_opnvwk(work_in,&handle,work_out);
  if (handle)
  {
    v_clrwk(handle);  /* clear screen */
    v_ellpie(100,100,30,20,0,1800);
    /* half an ellipse*/
    evnt_keybd();
    v_clsvwk(handle);
  }
  return appl_exit();
}
```

# v_ellipse — Draw an ellipse

*Class: VDI* *Category: GDP Output*

## SYNOPSIS

```
#include <vdi.h>

v_ellipse(handle,x,y,xradius,yradius);

int handle;              workstation handle
int x;                   x co-ordinates of centre
int y;                   y co-ordinate of centre
int xradius;             x radius of circle
int yradius;             y radius of circle
```

## DESCRIPTION

This 'Generalised Drawing Primitive' (GDP) is used to draw an ellipse using the fill area attributes (vsf_color etc.). The ellipse is drawn with centre (x, y) and of the given xradius and yradius in their native co-ordinates.

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_circle, vsf_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],junk,handle;
  appl_init();
  handle=graf_handle(&junk,&junk,&junk,&junk);
  v_opnvwk(work_in,&handle,work_out);
  if (handle)
  {
    v_ellipse(handle,100,100,30,30);
    /* draws a filled ellipse centred at (10,100),
       radius 30,30 pixels in black */
    evnt_keybd();
    v_clsvwk(handle);
  }
  return appl_exit();
}
```



# v_enter_cur, v_exit_cur — Enter/Exit alpha mode

*Class: VDI* *Category: Screen Escape Functions*

## SYNOPSIS

```
#include <vdi.h>

v_enter_cur(handle);
v_exit_cur(handle);

int handle;          workstation handle
```

## DESCRIPTION

v_enter_cur exits graphics mode and enters cursor (or alpha) mode. On the ST this clears the screen to colour 0, turns off the mouse cursor (as if by v_rmcur) and turns on the TOS cursor.

Note that when calling v_enter_cur function you should ensure that the user has released the left mouse button (by watching it via vq_mouse), otherwise the VDI will fail to notice its release after calling the function and will wait for it to be 'released' on calling v_exit_cur.

The converse function is v_exit_cur which exits alpha (or cursor) mode and enters graphics mode. On the ST this turns off the TOS cursor and turns the mouse cursor on. Note that it does not cause the screen to be updated. If running under the AES this would normally be done using the form_dial call with the parameter FMD_FINISH.

Note that these calls are usually used by a GEM application which wishes to run a TOS program.

## SEE

form_dial

# v_fillarea

Draw a filled area

*Class: VDI*

*Category: Graphics Output*

## SYNOPSIS

```
#include  <vdi.h>

v_fillarea(handle,n,pxyarray);

int  handle;              workstation  handle
int  n;                   number  of  vertices
short  *pxyarray;         co-ordinate  values
```

## DESCRIPTION

This function is used to plot a filled area. The vertices of the polygon to fill are passed in pxyarray with (pxyarray(0), pxyarray(1)) giving the first point, (pxyarray(2), pxyarray(3)) giving the second point etc.

Note that unlike the Line-A routine it is not necessary to specify the first point as the end point.

How the area is drawn depends on the fill area attributes (see vsf_interior etc.).

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

## SEE

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;   /* 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)
  {
    v_fillarea(handle,3,pts);
    /* draws  a  triangle  with  corners  at  (10,20),
        (100,40)  and  (20,100)  */
    v_clsvwk(handle);
  }
  return  appl_exit();
}
```



# v_font

Change default alpha text font

*Class: Lattice*

*Category: Atari Escape Functions*

## SYNOPSIS

```
#include  <vdi.h>

v_font(handle,font);

int  handle;      screen  workstation  handle
void  *font;      pointer  to  font  header
```

## DESCRIPTION

This function changes the default alpha text font (as used written by v_curtext and printf etc.). The font parameter must point to a Line-A font header (as given by the type LA_FONT in linea.h).

This function is most often used to give 8x8 characters (and thus 50 lines) on monochrome screens. This technique is used by Batcher.

This function is not officially documented but is implemented on all current versions of the operating system. Note that this means it cannot be guaranteed to work correctly in all circumstances.

## SEE

linea0, linea8

## EXAMPLE

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

int  main(void)
{
  int  handle;
  short  junk;

  appl_init();
  handle=graf_handle(&junk,&junk,&junk,&junk);
  linea0();
  v_font(handle,la_init.li_a1[1]);
  /* use  the  8x8  system  font  to  give  50  lines  on  mono
    displays  */
  return  appl_exit();
}
```

# v_form_adv

Printer form advance

*Class: VDI*

*Category: Printer Escape Functions*

## SYNOPSIS

```
#include <vdi.h>

v_form_adv(handle);

int handle;        workstation handle
```

## DESCRIPTION

This function advances to a new page and can only be used with a printer handle under GDOS.

You might use this function rather than v_clrwk if you wanted to draw a second page which included all the graphics on the current page.

## SEE

v_updwk, v_clrwk



# v_get_pixel

Return the pixel value of given point

*Class: VDI*

*Category: Raster Functions*

## SYNOPSIS

```
#include <vdi.h>

v_get_pixel(handle,x,y,pel,index);

int handle;        workstation handle
int x;             x co-ordinate of pixel
int y;             y co-ordinate of pixel
short *pel;        pixel value
short *index       corresponding colour index
```

## DESCRIPTION

This function is used to find the pixel value (or colour index) of the point (x, y) on the device specified by handle.

The function returns the pixel value of the point in pel, and the corresponding colour index value in index. For the mapping from pixels to colour indices see vr_trnfm.

Note that this function is normally only available on screen devices and is not required even then.

## SEE

vr_trnfm

# v_gtext

Draw graphics text

*Class: VDI*

*Category: Graphics Output*

## SYNOPSIS

```
#include <vdi.h>

v_gtext(handle,x,y,str);

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

## DESCRIPTION

This function is used to display text on the screen. The string to write is passed in str and it is displayed starting at position (x, y).

How the text is drawn depends on the text attributes (see vst_height). You can determine how big the text that you draw with v_gtext will be, by using the vqt_extent function. To draw justified text, use the v_justified function.

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

## SEE

vst_height, vswr_mode, vst_point, vst_rotation, vst_font, vst_color, vst_effects, vst_alignment, 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)
  {
    v_gtext(handle,20,20,"Hello World");
     /* writes hello world at 20,20 */
      .....
    v_clsvwk(handle);
  }
  return appl_exit();
}
```



# v_hardcopy

Copy screen to printer

*Class: VDI*

*Category: Screen Escape Functions*

## SYNOPSIS

```
#include <vdi.h>

v_hardcopy(handle);

int handle;         workstation handle
```

## DESCRIPTION

This function dumps the screen to the printer in the same form as with the Alt-Help key.

The workstation handle should be a screen workstation handle.

## SEE

Prtblk, Scrdmp

# v_hide_c

Hide mouse cursor

*Class: VDI*

*Category: Input Functions*

## SYNOPSIS

```
#include <vdi.h>

v_hide_c(handle);

int handle;                    workstation handle
```

## DESCRIPTION

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

v_hide_c will always hide the mouse. v_show_c can be used to display it once more.

## SEE

graf_mouse, v_show_c



# v_justified

Draw justified graphics text

*Class: VDI*

*Category: Graphics Output*

## SYNOPSIS

```
#include <vdi.h>

v_justified(handle,x,y,str,len,word,chr);

int handle;          workstation handle
int x;               x co-ordinate of start
int y;               y co-ordinate of start
const char *str;     characters to output
int len;             width of string in pixels
int word;            1= modify inter-word spacing
                     0= leave inter-word spacing
int chr;             1= modify inter-char spacing
                     0= leave inter-char spacing
```

## DESCRIPTION

This 'Generalised Drawing Primitive' (GDP) function is used to display justified text on the screen. The string to write is passed in str and is displayed starting at position (x, y) in a width of len pixels. 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.

If the word parameter is 1 then the VDI may attempt to adjust the inter-word spacing to fit the string in the given width. If the chr parameter is 1 then the VDI may attempt to adjust the inter-character spacing.

How the text is drawn depends on the text attributes (See vst_height etc.).

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

To draw 'ordinary' un-justified text, use the v_gtext function.

## SEE

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

## 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)
  {
    v_justified(handle,20,20,"hello  ",70,1,1);
    /* writes hello world at 20,20 */
    evnt_keybd();
    v_clsvwk(handle);
  }
  return  appl_exit();
}
```



# v_meta_extents

Set the metafile binding rectangle

*Class: VDI*

*Category: Metafile Escape Functions*

## SYNOPSIS

```
#include  <vdi.h>

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

int  handle;      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 lets you set the extent rectangle in the metafile header. This informs other programs of a rectangle in which the metafile graphics will fit. If this function is not called then zeroes will be written to the appropriate place in the metafile, indicating an indeterminate size.

The (min_x, min_y) and (max_x, max_y) co-ordinates give the bounding rectangle.

## SEE

v_opnwk, vm_pagesize, vm_coords

VDI Escape 101

# v_offset

Change console screen offset

*Class: Lattice*

*Category: Atari Escape Functions*

## SYNOPSIS

```
#include <vdi.h>

v_offset(handle,lines);

int handle;     screen workstation handle
int lines;      y co-ordinate in pixels
```

## DESCRIPTION

This function changes the origin of the console screen (as used written by v_curtext and printf etc.). The lines parameter gives the y co-ordinate of the top of the new screen.

After calling this function you should clear the screen (via v_clrwk) to re-initialise the system's internal variables. If you call this function then the screen will not normally scroll correctly unless you modify the Line-A variables correctly.

This function is not officially documented but is implemented on all current versions of the operating system.

## SEE

linea0

Input parameters

contrl (0) = Opcode 5
contrl (1) = Number of points in PTSIN array (0)
contrl (3) = Lenght of the INTIN array (1)
contrl (5) = 101, 101
contrl (6) = handle

intin (0) = y co-ordinate in pixels

Antal linjer + bredden på skärmen i bytes.



# v_opnvwk

Open virtual workstation

*Class: VDI*

*Category: Workstation Control*

## SYNOPSIS

```
#include <vdi.h>

v_opnvwk(work_in, handle, work_out);

short *work_in;         input parameters
short *handle;          workstation handle
short *work_out;        output characteristics
```

## DESCRIPTION

This function is used to open a virtual workstation and can be used regardless of whether GDOS is loaded. The work_in and work_out parameters are the same as for v_opnwk, the open physical workstation call.

The handle parameter is different, however. On input, it must point to a variable giving the physical handle of the device. After the v_opnvwk call, it will be updated to contain a virtual workstation handle that can be used for subsequent VDI calls.

You should obtain the physical workstation handle for the screen from the graf_handle AES call, as shown below.

Use device number 1 for the screen in work_in(0) when not using GDOS. This function will return 0 in handle if the virtual workstation cannot be opened. If the call is successful then you must call v_clsvwk before your program terminates.

If you wish to use GDOS the device number passed in work_in(0) should be 2 + Getrez() (from the XBIOS). This will ensure that the right fonts are obtained for the current screen mode.

If you wish you can open more than one virtual workstation on the same device. This enables you to switch between different settings for line or fill styles without using any VDI calls.

## SEE

v_opnwk, v_clswvk, graf_handle

## 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)
  {
    /* Now the main program */
    .....
    v_clsvwk(handle);
  }
  return  appl_exit();
}
```



# v_opnwk

Open physical workstation

*Class: VDI*

*Category: Workstation Control*

## SYNOPSIS

```
#include  <vdi.h>

v_opnwk(work_in,  handle,  work_out);

short  *work_in;          input  parameters
short  *handle;           new  workstation  handle
short  *work_out;         output  characteristics
```

## DESCRIPTION

This function is used to open a physical workstation and can only be used with GDOS present. To check for the presence of GDOS use the vq_gdos function. The work_in parameter should contain 11 shorts as follows:

| | |
|---|---|
| work_in(0) | Device identification number. This gives the device driver to load according to the ASSIGN.SYS file. |
| work_in(1) | Line type. |
| work_in(2) | Line colour index. |
| work_in(3) | Marker type. |
| work_in(4) | Marker colour index. |
| work_in(5) | Text face. |
| work_in(6) | Text colour index. |
| work_in(7) | Fill interior style. |
| work_in(8) | Fill style index. |
| work_in(9) | Fill colour index. |
| work_in(10) | NDC to RC transformation flag:<br>0 = Use NDC (normalised device co-ordinates) i.e. each page has co-ordinates 0 to 32767 regardless of the physical screen size.<br>1 = Reserved.<br>2 = Use RC (raster co-ordinates) e.g. physical screen co-ordinates. |

The values for work(1) to work(9) are the initial values for the line, marker, text and fill attributes; use 1 for sensible defaults. Note that only RC co-ordinates are available with the standard ST screen drivers, GDOS is required to gain access to NDC co-ordinates.

The conventional values for device numbers are as follows:

| | |
|---|---|
| 1-9 | Screens |
| 11-20 | Plotters |
| 21-30 | Printers |
| 31 | Metafile |
| 41-50 | Cameras |
| 51-60 | Tablets |

handle is used in a similar manner to v_opnvwk, but the value on entry is ignored and the value returned in it is the handle to use when making further VDI calls for this device; thus by having separate printer and screen handles you can output to a printer and to the screen at the same time. If the device cannot be opened then 0 is returned in handle.

If the device was successfully opened then the work_out array (which must have enough room for 57 shorts) is filled out as follows:

| | |
|---|---|
| work_out(0) | Device width in pixels starting from 0. E.g. on medium and high resolution screens this is 639. |
| work_out(1) | Device height in pixels starting from 0. E.g. 199 for medium resolution screens and 399 for high resolution. |
| work_out(2) | Device co-ordinate units flag:<br>0 = capable of precisely scaled image.<br>1 = not capable of precisely scaled image. |
| work_out(3) | Width of one pixel in microns. |
| work_out(4) | Height of one pixel in microns. |
| work_out(5) | Number of character heights:<br>0 = continuous scaling. |
| work_out(6) | Number of line types. |



| | |
|---|---|
| work_out(7) | Number of line widths:<br>0 = continuous scaling |
| work_out(8) | Number of marker types. |
| work_out(9) | Number of marker sizes:<br>0 = continuous scaling |
| work_out(10) | Number of faces (fonts) supported. |
| work_out(11) | Number of patterns available. |
| work_out(12) | Number of hatch styles available. |
| work_out(13) | Number of predefined colours (e.g. 2 for monochrome, 4 for medium resolution). |
| work_out(14) | Number of Generalised Drawing Primitives (GDPs). |
| work_out(15)<br>to<br>work_out(24) | List of the first 10 supported GDPs. The number indicates which GDP. -1 indicates the end of the list. GEM VDI defines 10 GDPs:<br>1 Bar.<br>2 Arc.<br>3 Pie slice.<br>4 Circle.<br>5 Ellipse.<br>6 Elliptical arc.<br>7 Elliptical pie.<br>8 Rounded rectangle.<br>9 Filled rounded rectangle.<br>10 Justified graphics text. |
| work_out(25)<br>to<br>work_out(34) | List of the attribute set used with each GDP:<br>0 Polyline.<br>1 Polymarker.<br>2 Text.<br>3 Fill area.<br>4 None. |
| work_out(35) | Colour capability flag:<br>0 no.<br>1 yes. |

| | |
|---|---|
| work_out(36) | Text rotation capability flag:<br>0 no.<br>1 yes. |
| work_out(37) | Fill area capability flag:<br>0 no.<br>1 yes. |
| work_out(38) | Cell array operation capability flag:<br>0 no.<br>1 yes. |
| work_out(39) | Number of available colours in palette:<br>0 continuous device (>32767 colours).<br>2 monochrome .<br>>2 number of colours. |
| work_out(40) | Number of locator devices:<br>1 Keyboard only.<br>2 Keyboard and other input. |
| work_out(41) | Number of valuator devices:<br>1 Keyboard only.<br>2 Other valuator device is available. |
| work_out(42) | Number of choice devices:<br>1 function keys on keyboard.<br>2 if another keypad is available. |
| work_out(43) | Number of string devices:<br>1 keyboard. |
| work_out(44) | Workstation type:<br>0 output only.<br>1 input only.<br>2 input/output.<br>4 metafile output. |
| work_out(45) | Minimum character width in pixels. |
| work_out(46) | Minimum character height in pixels. |
| work_out(47) | Maximum character width in pixels. |



| | |
|---|---|
| work_out(48) | Maximum character height in pixels. |
| work_out(49) | Minimum line width. |
| work_out(50) | 0 |
| work_out(51) | Maximum line width. |
| work_out(52) | 0 |
| work_out(53) | Minimum marker width. |
| work_out(54) | Minimum marker height. |
| work_out(55) | Maximum marker width. |
| work_out(56) | Maximum marker height. |

## SEE

v_opnvwk, v_clswk, vq_gdos

## 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);
    if (handle)
    {
      /* Now write to the printer */
      .....
      v_clswk(handle);
    }
    else
      printf("Could not open printer");
  }
  else
    printf("Graphics on the printer needs GDOS");
  return 0;
}
```

# v_output_window

Write part of a page to printer

*Class: VDI*

*Category: Printer Escape Functions*

## SYNOPSIS

```
#include <vdi.h>

v_output_window(handle,pxyarray);

int handle;         workstation handle
short *pxyarray;    rectangle giving area to print
```

## DESCRIPTION

This function prints the part of the current page specified by (pxyarray(0), pxyarray(1)) to (pxyarray(2), pxyarray(3)). This can only be used with a printer handle under GDOS.

This is similar to v_updwk except that only the specified area is printed.

## SEE

v_updwk, v_clrwk



# v_pline

Draw one or more lines (polyline)

*Class: VDI*

*Category: Graphics Output*

## SYNOPSIS

```
#include <vdi.h>

v_pline(handle,n,pxyarray);

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

## DESCRIPTION

This function is used to plot a series of lines between 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.

Thus to draw a single line use n=2. This function can also be used to plot a single point with pxyarray(2)=pxyarray(0) and pxyarray(3)=pxyarray(1).

How the line is drawn depends on the line attributes (see vsl_type etc.).

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

## SEE

vsl_type, vswr_mode, vsl_udsty, vsl_width, vsl_color, vsl_ends

## 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_pline(handle,2,pts); /* draws a line between
                              (10,20) and (30,40) */
    v_clsvwk(handle);
  }
  return appl_exit();
}
```