Lattice C 5

the C Compiler for your Atari ST Computer

Volume III

Atari Library Manual

Requires:

- ✓ Atari 520ST upwards
  (1M+ memory advised)
- ✓ Disk drive
  (2 floppies or hard disk advised)
- ✓ Mouse

HiSoft
High Quality Software

Lattice C

The C system for your Atari ST

Volume III
Atari Library Manual

Published by HiSoft

Version 5
First edition March 1990 (ISBN 0 948517 31 X)
Second edition April 1991

ISBN for this volume 0 948517 39 5

ISBN for complete 3 volume set 0 948517 28 X

Set using an Apple Macintosh™ and Laserwriter™ with Microsoft Word™ and SuperPaint™.

Table of Contents

1 Introduction

This volume describes the Atari ST specific parts of the Lattice C library, covering the application environment services (AES), virtual device interface (VDI), graphics environment manager disk operating system (GEMDOS), basic input/output system (BIOS), extended basic input/output system (XBIOS) and Line-A functions. This gamut of functions is known collectively as the operating system (TOS).

The following sections provides detailed descriptions of the operating system functions often with examples and lists of known problems. All functions are described in the same basic way, with a synopsis, a description of the function as implemented, the input and output parameters and any side effects of the call, and finally any cross-references to other functions which are related or perform similar functions.

The synopses give a brief summary, listing the header file in which the function is declared, the calling syntax and the types of the parameters.

The calling form is listed as a one line summary, for instance form_center is:

```
#include <aes.h>

res=form_center(tree,x,y,w,h);
```

so that the function takes five parameters tree, x, y, w and h returning a single parameter. If the function does not return a value (i.e. 'returns void') then this is indicated by the return value not being assigned.

The type of parameters is then listed; note that the types listed are those used in the *definition*, to call them only compatible types are required. Hence considering form_center, the parameters are:

```
int res;            reserved
OBJECT *tree;       object tree to centre
short *x;           x co-ordinate of centred form
short *y;           y co-ordinate of centred form
short *w;           width of centred form
short *h;           height of centred form
```

So that the first parameter is a pointer to an object tree, and the second, third, fourth and fifth are pointers to variables which are to be filled in with the required co-ordinates. Note that in general these parameters would be passed as the address of a suitable variable.

Considering a more complex function such as vex_butv, the synopsis is:

```
#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
```

So that vex_butv takes three parameters and returns no value. Examining the types of the parameters, the first has type int. The next parameter is of type int (*)(short) i.e. a pointer to a function taking a single short parameter returning an int. Under older K&R compilers it was necessary to take the address of a function prior to passing as a parameter, however ANSI compilers will automatically perform this indirection, hence an explicit (&) is not needed. The final parameter is the address of a variable to be used to hold the vector and has type int (**)(short). For this a variable of type int (*)(short) would be declared and its address passed.

The final form which appears in the synopses are for the Line-A functions which usually take their parameters in the external Line-A parameter block. For instance the linea1 (plot pixel) function synopsis is:

```
#include  <linea.h>

linea1()

INTIN[0]=colour;              colour of pixel to plot
PTSIN[0]=X;                   X co-ordinate of pixel
PTSIN[1]=Y;                   Y co-ordinate of pixel
```

Hence linea1 takes no parameters and returns none, however three items in the Line-A parameter block must be set, INITIN(0), PTSIN(0) and PTSIN(1). These variables must be initialised prior to the call with the colour of the pixel, the X co-ordinate and the Y co-ordinate. Note that these Line-A variables exist in a private OS structure and must be accessed through several indirections hence various macros are provided.

The fonts used throughout this library manual are:

OCRB	Program fragments and synopses.
Avante Garde	Library identifiers, parameters, disk files and keyboard shortcuts. Note that square brackets (i.e. those used in array accesses) appear as ( ) in this font, whereas parentheses (i.e. those used in function calls) appear as ( ). Beware of the distinction.

Note that *italics* are used solely for emphasis.

2 AES Library

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

The AES provides the iconic user interface on the ST, dealing with resource files, objects, trees, dialog boxes and menus. It does not deal directly with the lower levels of the OS but communicates via the VDI.

The functions all communicate with the OS via several arrays, the most useful of these to the user is the global array, named _AESglobal. The elements of this are:

_AESglobal(0)	AES version number in major minor form.
_AESglobal(1)	Number of concurrent applications the AES supports (1 in all current versions).
_AESglobal(2)	Application identifier for this application (as returned by appl_init.
_AESglobal(3-4)	User global, a longword global available for use by the user.
_AESglobal(5-6)	Pointer to base of resource file loaded as the result of a rsrc_load call.
_AESglobal(7-14)	Reserved.

In general the functions provided are those available directly from the OS and use the standard ST names, however several functions have been added to give extra flexibility or functionality. These are: objc_walk, objc_xywh, rc_constrain, rc_copy, rc_equal, rc_inside, rc_intersect, rc_union, wind_info, wind_newdesk, wind_redraw and wind_title.

The current versions of the OS return the following AES version numbers:

Major	Minor	Name
1	20	ROM TOS (1.0), Blitter TOS (1.2)
1	30	Rainbow TOS (1.4), STE TOS (1.6)

It is best to check the AES version number when asking for a particular feature since an older version of TOS may be patched to include these features.

appl_exit

Exit application

Class: AES *Category: Application Control*

SYNOPSIS

```
#include <aes.h>

error=appl_exit();

int error;              return code
```

DESCRIPTION

This function should be called before a GEM AES application terminates, so that the AES may notice that it has finished. This does not terminate the program and should not be called unless appl_init has been called successfully.

Using this call causes AC_CLOSE messages to be sent to *all* desk accessories; note that this may include inactive ones, so a desk accessory should be prepared to ignore redundant AC_CLOSE messages.

RETURNS

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

SEE

appl_init

appl_find

Find an application's identifier

Class: AES *Category: Application Control*

SYNOPSIS

```
#include <aes.h>

ap_id=appl_find(name);

int ap_id;                    application identifier
const char *name;             name of application to find
```

DESCRIPTION

This function finds the application identifier of the application called name. This is the file name of the desk accessory whose identifier is being found. This must be 8 characters long, padded with spaces if required.

This is usually used in conjunction with appl_write to send a message to a desk accessory.

RETURNS

The function returns the application identifier that was requested or -1 if the application could not be found.

SEE

appl_init, appl_write, menu_register

EXAMPLE

```
#include <aes.h>

int main(void)
{
  int ap_id=appl_init();
  int saved_id;

  saved_id=appl_find("SAVED!   ");
  /*
   * Now write some code to send the open message
   */
  ...
  appl_exit();
  return 0;
}
```

appl_init

Initialise application

Class: AES *Category: Application Control*

SYNOPSIS

```
#include <aes.h>

ap_id=appl_init();

int ap_id;  application identifier
```

DESCRIPTION

This function should be called before calling any other GEM AES functions. It sets up some global areas that are used by the AES *and* the bindings to the AES, hence this call must be made for the bindings to function correctly. If this call has been successfully made, the program should call appl_exit before terminating.

RETURNS

The application's global identifier is returned. This integer is needed when calling the menu_register and appl_read functions.

If the value returned is -1 then the program should terminate without making any further GEM AES calls (including appl_exit).

SEE

appl_exit, appl_read, menu_register

EXAMPLE

```
/*
 * print out public information from the global array
 */

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

int main(void)
{
  appl_init();
  printf("Version number = %d.%x\n", _AESglobal[0]>>8,
    _AESglobal[0]&0xff);
  printf("Concurrent process count = %d\n",
    _AESglobal[1]);
  printf("AES application id = %d\n",_AESglobal[2]);
  appl_exit();
  return 0;
}
```

appl_read

Read from message pipe

Class: AES *Category: Application Control*

SYNOPSIS

```
#include <aes.h>

error=appl_read(ap_id,length,message);

int  error;               error return
int  ap_id;               application identifier
int  length;              number of bytes to read
void *message;            address of message to read
```

DESCRIPTION

This function can be used to read length bytes into the memory pointed to by message from an application's message pipe. The application's identifier is supplied in the ap_id parameter; this is usually obtained from the result of the appl_init call.

Normally there is no need to do this directly as the evnt_mesag and evnt_multi routines can be used to read the standard AES 16 byte messages, such as those for menu selection or screen redraw. However, if you wish to send your own messages (for example between a co-operating desk accessory and main program), then you will need to use this function.

RETURNS

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

SEE

evnt_mesag, appl_read, appl_init

appl_tplay

Playback recording of user's actions

Class: AES *Category: Application Control*

SYNOPSIS

```
#include <aes.h>

error=appl_tplay(mem, num, scale);

int error;          error code
EVNTREC *mem;       stored actions to play back
int num;            number of user actions to playback
int scale;          playback speed
```

DESCRIPTION

This function 'plays back' a series of events that have (normally) been recorded using the appl_trecord function. The details of the EVENTREC structure are described under appl_trecord.

The scale parameter gives the speed from 1 to 10000 determining the speed at which GEM AES plays back the recording. 100 means play back at normal speed, 200 at double speed, 50 at half speed etc.

RETURNS

This function always returns 1, indicating that the operation was successful.

SEE

appl_trecord

appl_trecord

Record a sequence of user's actions

Class: AES

Category: Application Control

SYNOPSIS

```
#include <aes.h>

ret=appl_trecord(mem, num);

int ret;                    number of events recorded
EVNTREC *mem;               area to store actions
int num;                    number of actions to record
```

DESCRIPTION

This function records a series of user actions which may then be 'played back' using the appl_tplay function. The mem parameter will normally be an array with enough elements to store num events.

The EVNTREC structure is defined as:

```
typedef struct
{
  long ap_event;
  long ap_value;
} EVNTREC;
```

The ap_event field indicates the type of the event. The meaning of the ap_value field depends on which event occurs, as given in the table below:

ap_event	type of event	meaning of the ap_value field
0	timer event	elapsed time in system ticks (1/200s)
1	button event	low word: button state (1 if down) high word: number of clicks
2	mouse event	low word: X co-ordinate of mouse position high word: Y co-ordinate of mouse position
3	keyboard event	low word: key code of key typed high word: shift key state

RETURNS

The number of events recorded is returned; this will normally be equal to the number requested.

SEE

appl_tplay, evnt_timer, evnt_keybd, evnt_mouse, evnt_button, evnt_multi

EXAMPLE

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

int main(void)
{
  static EVNTREC x[100];
  int i,count;

  appl_init();
  /* start recording */
  count=appl_trecord(x,sizeof(x)/sizeof(EVNTREC));

  for (i=0; i<count; i++)
    printf("%ld->%lx\n",x[i].ap_event,x[i].ap_value);

  appl_exit();
}
```

appl_write

Write to message pipe

Class: AES *Category: Application Control*

SYNOPSIS

```
#include <aes.h>

error=appl_write(ap_id,length,message);

int error;                error return
int ap_id;                application identifier
int length;               number of bytes to write
void *message;            address of message to write
```

DESCRIPTION

This function is used to write a message of length bytes from address message to the application with identifier ap_id.

This may be used to send 'fake' redraw or menu events to your own program by using the ap_id that is returned by appl_init.

It may also be used to send messages between an application and a co-operating desk accessory. The appl_find function may be used to find the identifier of another application.

RETURNS

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

SEE

appl_find, appl_init, appl_read

EXAMPLE

```
/* send a redraw message to a window rectangle */
#include <aes.h>

int send_redraw(int wh, GRECT *p)
{
  short msg[8];

  msg[0]=WM_REDRAW;
  msg[1]=_AESglobal[2];      /* find my apps id */
  msg[2]=0;                  /* length = 16 + 0 */
  msg[3]=wh;                 /* window to redraw */
  msg[4]=p->g_x;             /* window rectangle */
  msg[5]=p->g_y;
  msg[6]=p->g_w;
  msg[7]=p->g_h;
  return appl_write(msg[1],sizeof(msg),msg);
}
```

evnt_button

Wait for a mouse button state

Class: AES *Category: Event Handling*

SYNOPSIS

```
#include <aes.h>

clicks=evnt_button(maxclicks,mask,state,x,y,
                                  button,kstate);

int clicks;         the number of clicks that occurred
int maxclicks;      maximum number of clicks to wait for
int mask;           which buttons to wait for
int state;          the button state to wait for
short *x;           x-coordinate of the mouse
short *y;           y-coordinate of the mouse
short *button;      final mouse button state
short *kstate;      shift key status
```

DESCRIPTION

This function waits for a particular mouse button state. Button events may be used to detect single, or multiple clicks on either of the mouse buttons. To detect more than one event at once, the evnt_multi function must be used.

The maxclicks parameter gives the maximum number of clicks to wait for. To wait for both single and double clicks, use 2 for this parameter. The function will then return 2 if the user double-clicked or 1 if the user single clicked.

The mask parameter gives the mouse buttons that the application is interested in. This is a bitmap with bit 0 indicating the left mouse button and bit 1 the right mouse button. Thus if the program is only interested in the state of the left mouse button, use a mask parameter of 1.

The state parameter is the state that is being waited for; again this is a bitmap with a bit of 1 indicating that the button is down and a bit of 0 indicating that the button is up. For the usual case of the left button being down, this parameter should have a value of 1.

The final position of the mouse, when the function returns, is given in x and y. These co-ordinates are given in pixels relative to the top left hand corner of the screen.

The button parameter gives the final state of the mouse buttons, in a similar form to that used by the mask parameter.

The kstate parameter gives the final state of the shift keys depressed; again this is a bitmap with the following meanings:

Name	Value	Meaning
K_RSHIFT	0x0001	Right shift key depressed.
K_LSHIFT	0x0002	Left shift key depressed.
K_CTRL	0x0004	Ctrl key depressed.
K_ALT	0x0008	Alt key depressed.

In general, it is recommended that only the left button is used.

Note that although this function can be used to wait for a click on the right hand button (mask=2, state=2) and for both buttons being clicked at once (mask=3, state=3) and to ensure that both buttons are not pressed (e.g. mask=3, state=1 waits for the left button only to be pressed); it can not be used to wait for a click on *either* the left *or* right buttons. See the VDI example for vex_butv to see how to detect a click on either button.

RETURNS

The function returns the number of mouse clicks which occurred.

SEE

evnt_multi

evnt_dclick

Get/Set the double-click speed of the mouse

Class: AES *Category: Event Handling*

SYNOPSIS

```
#include <aes.h>

res=evnt_dclick(new,flag);

int res;              new double click speed
int new;              the new mouse speed (0-4)
int flag;             if 1 set the speed,
                      if 0 get the speed
```

DESCRIPTION

This function either reads the current mouse double click speed or sets it to a new value. The values are the same as used by the Control Panel with 0 corresponding to the slowest and 4 to the fastest. If flag is 0 then the value of new is ignored. Note that the double click speed should only be altered at the request of the user and *not* at the whim of the programmer.

RETURNS

The return value of this function is the new double click speed (i.e. the old speed if the value was not changed).

evnt_keybd

Wait for keyboard event

Class: AES *Class: Event Handling*

SYNOPSIS

```
scancode=evnt_keybd();

int  scancode;     key  pressed
```

DESCRIPTION

This function waits for a key to be pressed or returns a key that has been pressed, but not yet returned to the program.

To detect more than one event at once, the evnt_multi function must be used.

RETURNS

The bottom eight bits returned are the ASCII code for the character. The top eight bits are the scan code for the key. This enables non-ASCII keys such as the cursor control and function keys to be detected.

Although the high byte is normally the scan code, it is not when Ctrl is held down when the cursor left, cursor right and Clr/Home keys are pressed, in which case 0x73, 0x74 and 0x77 respectively are returned. Note that the scan codes for keys differ on machines which are nationalised for different countries. You should consult the XBIOS keyboard maps (see Keytbl) to obtain consistent keycodes across different keyboards.

SEE

evnt_multi, Keytbl

evnt_mesag

Wait for a message event

Class: AES *Category: Event Handling*

SYNOPSIS

```
#include <aes.h>

ret=evnt_mesag(msg);

int ret;               error code
short *msg;            message buffer
```

DESCRIPTION

This function returns the next message event. The msg parameter is usually an array of 8 shorts whose elements are as follows:

msg(0)	The message type.
msg(1)	The application identifier of the application that sent the message. See appl_init and appl_find.
msg(2)	The length of the message not including the pre-defined 16 bytes. If this is greater than zero then this is a user-defined message and appl_read can be used to read the remainder of the message.

The remainder of the elements depend on the message type which may be one of the following:

MN_SELECTED	The user has selected a menu item: msg(3) the object number of the menu title selected msg(4) the object number of the menu item selected. These values are as supplied by the header file created by WERCS.
WM_FULLED	This message is sent to your program when the user clicks on a window's full box, indicating that the application should make the window as large as possible, or if it is already as large as possible, to return it to its previous size. You should use the WF_FULLXYWH, WF_PREVXYWH and WF_CURRXYYH parameters of wind_get and wind_set to help you implement this: msg(3) the handle of the window that is to be fulled.

WM_REDRAW	This message is sent by the AES when an area of the screen which is wholly or partially covered by one of your windows needs to be redrawn: msg(3) the handle of the window to redraw msg(4) the x co-ordinate of the area to be redrawn msg(5) the y co-ordinate of the area to be redrawn msg(6) the width of the area to be redrawn msg(7) the height of the area to be redrawn The rectangle given by the AES, will probably contain an area outside the work area of your window. As a result you should use the rc_intersect function to find out the area that you really need to update. See rc_intersect for an example.
WM_ARROWED	This message is sent to your program when the user manipulates the scroll parts of a window: msg(3) the handle of the window msg(4) the action requested: WA_UPPAGE page up (i.e. above the vertical scroll bar) WA_DNPAGE page down (i.e. below the vertical scroll bar) WA_UPLINE line up (i.e. the up arrow) WA_DNLINE line down (i.e. the down arrow) WA_LFPAGE page left (i.e. to the left of the horizontal scroll bar) WA_RTPAGE page right (i.e. to the right of the horizontal scroll bar) WA_LFLINE character left (i.e. the left arrow) WA_RTLINE character right (i.e. the right arrow) You should use the WF_HSLIDE and WF_VSLIDE parameters of wind_get and wind_set to help you implement these.
WM_HSLID	This message is sent when the user drags the slider of the horizontal scroll bar: msg(3) the handle of the window msg(4) the new position of the slider between 0 and 1000. 0 is the far left, 1000 is the far right. You can use wind_set with a parameter of WF_HSLIDE to help you implement this.

WM_VSLID	This message is sent when the user drags the slider of the vertical scroll bar: msg(3) the handle of the window msg(4) the new position of the slider between 0 and 1000. 0 is the top, 1000 is the bottom. You can use wind_set with a parameter of WF_VSLIDE to help you implement this.
WM_MOVED	This message is used to tell your program that the user has requested that the window be moved by dragging on the window's title bar: msg(3) the handle of the window msg(4) the x co-ordinate of the new window msg(5) the y co-ordinate of the new window msg(6) the new window width (will be the same as the current window width) msg(7) the new window height (will be the same as the current window height) The window co-ordinates given are the full size of the entire window including the title, scroll bars etc. Thus giving the appropriate values to pass to wind_set with WF_CURRXYWH without alteration. Note that this message and WM_SIZED are usually handled by common code, since they pass identical information.
WM_TOPPED	This message is sent to your program when the user clicks on a window to indicate that the window is to become the top window. Normally you should call wind_set with a parameter of WF_TOP to let the AES move your window to the top: msg(3) the handle of the window You will still be sent this message if a window other than your own has become the top window; if your application only has one window, check to see if msg(3) is really your window handle!
WM_CLOSED	This message is sent to your program when the user has clicked on a window's close box: msg(3) the handle of the window to be closed.

WM_SIZED	This message is used to tell your program that the user has requested a new window size by dragging on the window's size box: msg(3) the handle of the window msg(4) the x co-ordinate of the new window (will be the same as the current window x co-ordinate) msg(5) the y co-ordinate of the new window (will be the same as the current window y co-ordinate) msg(6) the new window width msg(7) the new window height The window co-ordinates given are the full size of the entire window including the title, scroll bars etc. Thus giving the appropriate values to pass to wind_set with WF_CURRXYWH without alteration. Note that a redraw message is only sent by the AES after a wind_set call if the window size increases in either direction, or if a new part is uncovered. 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 itself.
AC_OPEN	This message is used to tell a desk accessory that the user has clicked on its menu item and so it should open: msg(3) the desk accessory menu identifier as returned by the menu_register call.
AC_CLOSE	This message is used to tell a desk accessory that the current application has been terminated. Note that you should *not* close *or* delete any windows which you had open, as the Desktop, or other shell, will have done this for you. If you do attempt to close your windows the Desktop may hang. msg(3) the desk accessory menu identifier as returned by the menu_register call.

RETURNS

The return value of this function is reserved. Currently 1 is always returned.

SEE

evnt_multi, wind_get, wind_set

EXAMPLE

```
/* skeleton AES message loop */

#include <aes.h>

void do_full(int wh)
{
  GRECT c,p,f;

  /* get current size */
  wind_get(wh,WF_CXYWH,&c.g_x,&c.g_y,&c.g_w,&c.g_h);
  /* get full size */
  wind_get(wh,WF_FXYWH,&f.g_x,&f.g_y,&f.g_w,&f.g_h);
  /* if full size == current size */
  if (rc_equal(&c,&f))
  {
    /* then get previous size */
    wind_get(wh,WF_PXYWH,&p.g_x,&p.g_y,&p.g_w,&p.g_h);
    /* if previous != full size */
    if (!rc_equal(&p,&f))
      /* then set current size to previous size */
      wind_set(wh,WF_CXYWH,p.g_x,p.g_y,p.g_w,p.g_h);
    /* else do nothing */
  }
  else
    /* else set current size to full size */
    wind_set(wh,WF_CXYWH,f.g_x,f.g_y,f.g_w,f.g_h);
}

/* dispatch events until we fail to recognise one */
int do_mesag(void)
{
  for (;;)
  {
    short msg[8];

    evnt_mesag(msg);
    switch (msg[0])
    {
      case WM_REDRAW:
        wind_redraw(msg[3],(GRECT *)&msg[4],draw);
        break;

      case WM_TOPPED:
        wind_set(msg[3],WF_TOP);
        break;

      case WM_FULLED:
        do_full(msg[3]);
        break;

      case WM_SIZED:
      case WM_MOVED:
        wind_set(msg[3],WF_CXYWH,msg[4],msg[5],
          msg[6],msg[7]);
        break;

      default:
        return msg[0];
    }
  }
}
```

evnt_mouse — Wait for the mouse to enter/leave a rectangle

Class: AES — *Category: Event Handling*

SYNOPSIS

```
#include <aes.h>

res=evnt_mouse(flag,x,y,width,height,mx,my,
                                   button,kstate);

int res;             reserved
int flag;            enter or leave flag
int x;               x co-ordinate of watched rectangle
int y;               y co-ordinate of watched rectangle
int width;           width of watched rectangle
int height;          height of watched rectangle
short *mx;           final x-coordinate of the mouse
short *my;           final y-coordinate of the mouse
short *button;       final mouse button state
short *kstate;       shift key status
```

DESCRIPTION

This function waits for the mouse to enter/leave a given screen area. This may be used to give a special mouse form over a particular area of the screen.

The flag parameter should be 1 to wait for the mouse to leave the given rectangle and 0 to wait for it to enter. The x, y, width and height parameters specify the rectangle to be watched. This is a standard AES rectangle i.e. expressed in pixels from the top left of the screen.

The final position of the mouse, when the function returns, is given in mx and my. These co-ordinates are given in pixels relative to the top left hand corner of the screen.

The button parameter gives the final state of the mouse buttons, with bit 0 set if the left button is depressed and bit 1 set if the right button is pressed. The kstate parameter gives the final state of the shift keys depressed; again this is a bitmap with the following meanings:

Name	Value	Meaning
K_RSHIFT	0x0001	Right shift key depressed
K_LSHIFT	0x0002	Left shift key depressed
K_CTRL	0x0004	Ctrl key depressed
K_ALT	0x0008	Alt key depressed

RETURNS

The return value is reserved; 1 is always returned at present.

SEE

evnt_multi

evnt_multi

Wait for a number of events at once

Class: AES *Category: Event Handling*

SYNOPSIS

```
#include <aes.h>

res=evnt_multi(flags,bmaxclicks,bmask,bstate,
          m1flag,m1x,m1y,m1w,m1h,
          m2flag,m2x,m2y,m2w,m2h,
          mes,
          locount, hicount,
          x,y,button,kstate,kreturn,breturn);

int  res;           the events that actually occurred
int  flags;         which events to wait for
int  bmaxclicks;    maximum number of clicks to wait
                    for
int  bmask;         which buttons to wait for
int  bstate;        the button state to wait for
int  m1flag;        enter/leave flag of first mouse
                    rectangle
int  m1x;           x co-ordinate of first watched
                    rectangle
int  m1y;           y co-ordinate of first watched
                    rectangle
int  m1w;           width of first watched rectangle
int  m1h;           height of first watched rectangle
int  m2flag;        enter/leave flag of second mouse
                    rectangle
int  m2x;           x co-ordinate of second watched
                    rectangle
int  m2y;           y co-ordinate of second watched
                    rectangle
int  m2w;           width of second watched rectangle
int  m2h;           height of second watched rectangle
short  *mes;        message buffer
int  locount;       lower 16 bits of time in
                    milliseconds
int  hicount;       upper 16 bits of time in
                    milliseconds
short  *x;          x-coordinate of the mouse
short  *y;          y-coordinate of the mouse
short  *button;     final mouse button state
short  *kstate;     shift key status
short  *kreturn;    scancode of key pressed
short  *breturn;    number of mouse clicks
```

DESCRIPTION

This function waits for one or more events to occur. It is almost always the heart of a GEM program. Fortunately most of the parameters are the same as for the other event handling functions.

Flags specifies which events the AES should wait for. It is a bitmap with masks as follows:

MU_KEYBD	Wait for a keyboard event; the scancode of the key pressed will be returned in the parameter kreturn.
MU_BUTTON	Wait for a mouse button event. The bmaxclicks, bmask and bstate parameters have the same meaning as for the evnt_button function and the x, y, button and kstate parameters will be returned with the appropriate parameters.
MU_M1	Indicates that the m1flag, m1x, m1y, m1w and m1h parameters will be used as a watched rectangle as with a corresponding evnt_mouse call. Again the x, y, button and kstate parameters will be returned with the appropriate parameters.
MU_M2	Indicates that the m2flag, m2x, m2y, m2w and m2h parameters will be used as a second watched rectangle as with the corresponding evnt_mouse call. Again the x, y, button and kstate parameters will be returned with the appropriate parameters. This gives significantly more power than is available with evnt_mouse as two rectangles may be watched at once.
MU_MESAG	Wait for message events. If this mask is included and a message event occurs then the message will be stored at the address pointed to mes, as for the evnt_mesag call. This mask is almost always included.
MU_TIMER	Wait for a timer event. The hicount and locount parameters are used as for evnt_timer. This can be used to implement a flashing cursor, for example.

RETURNS

evnt_multi returns a mask with the same bit usage as the flags parameter indicating which events occurred. More than event can be returned at once, so ensure that your code handles this correctly or your program will 'miss' events.

SEE

evnt_keybd, evnt_button, evnt_mouse, evnt_mesag, evnt_timer

evnt_timer

Wait for time to pass

Class: AES *Category: Event Handling*

SYNOPSIS

```
#include <aes.h>

res=evnt_timer(locount,hicount);

int locount;  lower 16 bits of time in milliseconds
int hicount;  upper 16 bits of time in milliseconds
```

DESCRIPTION

This function waits for a certain number of milliseconds to pass. The AES may also re-schedule so as to run a desk accessory, for example. This means that the time passed is a minimum time which the AES will wait for. Programs that perform long calculations may wish to call evnt_timer with a value of 0 so that the user may use desk accessories whilst the calculation is in progress.

To detect more than one event at once, the evnt_multi function must be used.

RETURNS

The return value of this function is reserved. At the moment 1 is always returned.

SEE

evnt_multi

form_alert — Display an alert box and wait for reply

Class: AES — *Category: Form Handling*

SYNOPSIS

```
#include <aes.h>

res=form_alert(default,alert);

int res;                    button selected by the user
int default;                default exit value
const char *alert;          the text of the alert
```

DESCRIPTION

This function displays an alert on the screen and lets the user interact with it. The default button is given by the default parameter and is 1 for the first button, 2 for the second, etc., or 0 if there is no default button. The screen is restored by the AES so there is no need to redraw the screen. alert has the form:

```
[icon][message][button1|button2.....]
```

icon is the number of the icon to display:

0	No icon
1	! icon
2	? icon
3	STOP icon

message is the text to display in the alert box; it should not exceed 200 characters and should contain | (vertical bar) characters to delimit the lines (of which there may be at most 5), the text of which should not exceed 30 characters per line. button1 and button2 are the text for the buttons. There may be up to three buttons; the text for each cannot exceed 10 characters.

Under TOS 1.0, if the width of all the buttons is wider than the text then the buttons are moved to the right, so that some of the buttons are inaccessible. This can be avoided by padding one of the lines with spaces if you have a particularly wide button set. This *only* works if you also have an icon.

On TOS 1.2 and above there is a different problem, if you have an icon-less alert and your text is longer than the buttons then the last character of the long line will impinge on the right-hand border of the alert. This can be avoided by adding a space on to the longest line in icon-less alerts.

If the text parameter does not conform to the above rules the machine may crash.

RETURNS

The value returned is the number of the button selected.

SEE

objc_draw, form_dial, form_do

EXAMPLE

```
/*
 * initialise memory block for file
 */

#include <aes.h>
#include <stdlib.h>
#include <stdio.h>
#include <limits.h>

void *load_file(FILE *fp)
{
  void *p;

  /* get memory */
  p=malloc(filelength(fileno(fp)));
  if (!p)
    form_alert(1,"[3][Out of memory][OK]");
  else
    fread(p,1,LONG_MAX,fp);     /* read whole file */
  return p;
}
```

form_button

Dialog handler mouse primitive

Class: AES *Category: Form Handling*

SYNOPSIS

```
#include  <aes.h>

res=form_button(tree,obj,clicks,newobj);

int res;                  exit condition flag
OBJECT *tree;             form being handled
int obj;                  current editable object
int clicks;               number of clicks
short *newobj;            next object
```

DESCRIPTION

This function need only be used when writing your own form handler to replace form_do. It is used to handle the mouse clicks which control the location of text to be entered and changes in button states.

The value tree contains a pointer to the current object tree being manipulated, and obj the object currently being edited. The clicks parameter gives the number of clicks which the application received. form_button processes this information to produce a value for newobj giving the next object which is to be edited. Note that the top bit of newobj will be set if an exit object was double clicked.

RETURNS

The function returns the value 0 if an object which had the EXIT or TOUCHEXIT bits set was selected. Otherwise the value 1 is returned.

SEE

form_do, form_keybd, objc_edit

EXAMPLE

```
/*
 *  Implement our own version of form_do
 *
 *  the starting object number must be valid
 */

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

int my_form_do(OBJECT *tree, short next)
{
  short edit;
  short which, cont;
```

```
    short  idx;
    short  x, y, kr, br;
    short  junk;

    wind_update(BEG_UPDATE);
    edit=0;
    cont=1;
    while (cont)
    {
      /* position the cursor on an editing field */
      if (next!=0 && edit!=next)
      {
        edit = next;
        next = 0;
        /* turn on the text cursor and initialise idx */
        objc_edit(tree, edit, 0, &idx, ED_INIT);
      }
      /* wait for mouse or key */
      which=evnt_multi(MU_KEYBD | MU_BUTTON,
       0x02, 0x01, 0x01,
       0, 0, 0, 0, 0,
       0, 0, 0, 0, 0,
       NULL,
       0, 0,
       &x, &y, &junk, &junk, &kr, &br);
      if (which & MU_KEYBD)
      {
        /* process the keystroke */
        cont=form_keybd(tree, edit, 0, kr, &next, &kr);
        if (kr)
          /* if not special then edit the form */
          objc_edit(tree, edit, kr, &idx, ED_CHAR);
      }
      if (which & MU_BUTTON)
      {
        /* find the object under the rodent */
        next=objc_find(tree, ROOT, MAX_DEPTH, x, y);
        if (next==NIL)
        {
          /* If no object then ring the bell */
          Bconout(2,'\a');
          next = 0;
        }
        else
          /* else process the button */
          cont=form_button(tree, next, br, &next);
      }
      /* If finished or moving to a new object */
      if (!cont || (next!=0 && next != edit))
        /* then hide the text cursor */
        objc_edit(tree, edit, 0, &idx, ED_END);
    }
    wind_update(END_UPDATE);
    return next;
}
```

form_center

Centre a dialog box on the screen

Class: AES *Category: Form Handling*

SYNOPSIS

```
#include <aes.h>

res=form_center(tree,x,y,w,h);

int res;                reserved
OBJECT *tree;           object tree to centre
short *x;               x co-ordinate of centred form
short *y;               y co-ordinate of centred form
short *w;               width of centred form
short *h;               height of centred form
```

DESCRIPTION

This function centres the dialog box at address tree on the screen. This function is normally used before calling objc_draw to display a form. The call modifies the root object of the form and also returns the centred values in x, y, w and h ready for use with objc_draw; note that these values include the width of any border or outline specified by the root object and so may be a larger rectangle than that given in the object definition.

RETURNS

The function return value is reserved; it is always 1 at present.

SEE

objc_draw, form_do

EXAMPLE

```
/*
 * generalised form set up routine, find the tree
 * and then centre it, returning a pointer to it.
 */

#include <aes.h>

OBJECT *start_form(int form, GRECT *p)
{
  OBJECT *tree;

  rsrc_gaddr(R_TREE,ROOT,&tree);    /* find a tree */
  form_center(tree,&p->g_x,&p->g_y,&p->g_w,&p->g_h);
  return tree;
}
```

form_dial — Dialog control function

Class: AES — *Category: Form Handling*

SYNOPSIS

```
#include <aes.h>

res=form_dial(flag,x1,y1,w1,h1,x2,y2,w2,h2);

int res;        error return
int flag;       operation to perform
int x1;         x co-ordinate of smaller rectangle
int y1;         y co-ordinate of smaller rectangle
int w1;         width of smaller rectangle
int h1;         height of smaller rectangle
int x2;         x co-ordinate of larger rectangle
int y2;         y co-ordinate of larger rectangle
int w2;         width of larger rectangle
int h2;         height of larger rectangle
```

DESCRIPTION

This function performs a number of operations concerned with dialog boxes according to the value of flag:

FMD_START	Should be called before a series of form_dial calls, although this does nothing on current versions of the operating system. This call is used to reserve the screen area inside the rectangle given by x2, y2, w2, h2.
FMD_GROW	Draws a box expanding from the rectangle given by x1, y1, w1, h1 to the rectangle given by x2, y2, w2, h2.
FMD_SHRINK	Draws a box shrinking from the rectangle given by x2, y2, w2, h2 to the rectangle given by x1, y1, w1, h1.
FMD_FINISH	Sends messages to re-draw the screen for any windows inside the rectangle given by x2, y2, w2, h2. If your application has displayed the form on top of one its windows, ensure that you respond to WM_REDRAW messages (see evnt_mesag), otherwise the dialog box will still be displayed on the screen.

RETURNS

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

SEE

objc_draw, form_do

EXAMPLE

```
/*
 * initialise a form ready for drawing.
 * starts by getting a form using start_form
 * (from form_center) and then reserves and
 * zooms.
 */

#include <aes.h>

OBJECT *init_form(int obj)
{
  GRECT p;
  OBJECT *tree;

  /* get a pointer to the object given by obj */
  tree=start_form(obj,&p);
  /* reserve the screen area */
  form_dial(FMD_START,0,0,0,0,
    p->g_x,p->g_y,p->g_w,p->g_h);
  /* draw a zoom box from the centre outwards */
  form_dial(FMD_GROW,
    p->g_x+p->g_w/2,p->g_y+p->g_h/2,0,0,
    p->g_x,p->g_y,p->g_w,p->g_h);
  return tree;
}
```

form_do

Let the user fill in a form

Class: AES *Category: Form Handling*

SYNOPSIS

```
#include <aes.h>

res=form_do(tree,startob);

int res            exit object index
OBJECT *tree;      object tree of the form
int startob;       editable object to start with
```

DESCRIPTION

This function is used to let the user fill in a form or dialog box. The tree parameter is the address of the form and is normally as found from rsrc_gaddr. The AES needs to know which editable text item to display the initial text cursor. This should be passed as the startob parameter. If there are no editable text fields, or you wish to start editing at the first editable field then the value 0 should be used.

The form should be drawn using objc_draw before calling this function.

RETURNS

This function returns the object index of the item that caused the dialog to finish (e.g. that of an OK button). Your program can then compare this with the values in the resource header file. Note that the value returned may be negative indicating that the exit object was double clicked in which case the bottom 15 bits should be masked off to find the true exit button. Also the exit object is not automatically de-selected when form_do returns, so you should do this manually.

SEE

objc_draw, form_center, form_dial

EXAMPLE

```
#include <aes.h>
void do_form(int obj,int *res)
{
  OBJECT *tree;

  tree=show_form(obj);      /* display a form */
  *res=form_do(tree,0);     /* interact with form */
  /* de-select the exit object */
  tree[*res&0x7fff].ob_state&=~SELECTED;
  clean_form(tree);    /* release the screen area */
}
```

form_error

Display a GEMDOS error alert

Class: AES *Category: Form Handling*

SYNOPSIS

```
#include <aes.h>

res=form_error(num);

int res;   button selected by the user
int num;   'PCDOS' error code
```

DESCRIPTION

This function displays a GEMDOS error message on screen. Unfortunately the routine does not take a GEMDOS error number, but a 'PCDOS error code' and it only produces messages for some error numbers. This number is passed in the num parameter.

The error numbers that form_error recognises are as follows:

2, 3, 18	This application cannot find the folder or file that you tried to access.
4	This application does not have room to open another document. To make room, close any document that you do not need.
5	An item with this name already exists in the directory, or this item is set to read-only status.
8, 10, 11	There is not enough memory for the application you just tried to run.
15	The drive you specified does not exist.

See the example below to display an error alert given that a GEMDOS error has occurred.

RETURNS

Theoretically this function could return a value different from 1, i.e. the exit button used, but as there is only ever one button displayed this is not possible.

SEE

form_alert

EXAMPLE

```
/*
 * display an error message based on the last
 * GEMDOS error encountered by the run-time
 * support library
 */

#include <aes.h>
#include <dos.h>

void error(void)
{
  graf_mouse(ARROW, NULL);
  if (_OSERR < 50)
    _OSERR -= 31;
  form_error(_OSERR);
}
```

form_keybd — Dialog handler keyboard primitive

Class: AES *Category: Form Handling*

SYNOPSIS

```
#include <aes.h>

res=form_keybd(tree,obj,nextobj,keyin,newobj,outkey);

int res;           exit condition flag
OBJECT *tree;      form being handled
int obj;           current editable object
int nextobj;       reserved; use the value 0
int keyin;         key whose action is to be performed
short *newobj;     next object
short *outkey;     modified key
```

DESCRIPTION

This function need only be used when writing your own form handler to replace form_do. It is used to handle the keys such as Return, Tab and the cursor keys.

The tree and obj parameters give an object tree and and the number of the object currently being edited. The value of keyin is the that obtained from the AES after an evnt_keybd (or evnt_multi) which form_keybd is to process.

The value returned in newobj is the object which is to be the next editable object if one of the special keys was used, or the exit object if Return was pressed and a default object existed. The value in outkey is the modified key stroke ready for passing to objc_edit, or zero if the key stroke was processed by form_keybd (i.e. was one of the special keys).

RETURNS

The value returned is zero if the processing of the key stroke caused an exit condition to occur, i.e. Return was pressed and a default exit object existed, otherwise the value returned is 1.

SEE

form_button, objc_edit

EXAMPLE

See form_button for an example of form_keybd.

fsel_exinput

Get a file name using the extended file selector

Class: AES *Category: File Selector Handling*

SYNOPSIS

```
#include <aes.h>

res=fsel_exinput(path,file,button,label);

int res;                    error return
char *path;                 directory displayed/chosen
char *file;                 file displayed/chosen
short *button;              the exit button the user
                            selected
const char *label;          title to display
```

DESCRIPTION

This function displays and lets the user interact with the extended GEM file selector, whilst displaying a message to indicate the action about to be taken (e.g. Save File).

The parameters of this call are the same as for fsel_input except for the extra label parameter. This string (which may be up to 30 characters long) is displayed instead of the Item Selector message.

The initial folder is specified by the path parameter; this will be updated by the call to give any new directory selected by the user. Similarly the file parameter gives the initial value for the file name selected and this will change if the user selects another file. The path buffer should be FMSIZE characters long and the file name FNSIZE characters long. Both these constants are defined in the dos.h header file.

The button parameter is returned as 1 if the user selects OK (or presses Return) or 0 if the user selects Cancel.

Note that this operating system call was added in AES version 1.30 (Rainbow TOS). However the binding in Lattice C will also work on earlier versions of the OS, displaying a box above the standard file selector.

RETURNS

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

SEE

fsel_input

EXAMPLE

```
/*
 * present a standard file selector
 */

#include <stdio.h>
#include <string.h>
#include <aes.h>
#include <dos.h>

int loadfile(void)
{
  static char select[FNSIZE];
  static char dirname[FMSIZE];
  short button;

  getcd(0,dirname);   /* get current directory */
  strcat(dirname,"\\*.*");
  *select=0;  /* start with an emtpy name */
  /* call fsel_exinput, always safe in Lattice 5 */
  fsel_exinput(dirname,select,&button,
   "Load A File");

  if (button)
    /* user selected file */
  else
    /* user cancelled */
}
```

fsel_input

Get a file name from the user using the file selector

Class: AES *Category: File Selector Handling*

SYNOPSIS

```
#include <aes.h>

res=fsel_input(path,file,button);

int res;                    error return
char *path;                 directory displayed/chosen
const char *file;           file displayed/chosen
short *button               the exit button the user
                            selected
```

DESCRIPTION

This function displays and lets the user interact with the standard GEM file selector.

The initial folder is specified by the path parameter; this will be updated by the call to give any new directory selected by the user. Similarly the file parameter gives the initial value for the file name selected and this will change if the user selects another file. The path buffer should be FMSIZE characters long and the file name FNSIZE characters long. Both these constants are defined in the dos.h header file.

The button parameter is returned as 1 if the user selects OK (or presses Return) or 0 if the user selects Cancel.

In general, we recommend that fsel_exInput is used rather than this function, because it has the advantage of informing the user of the action about to be taken.

RETURNS

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

SEE

fsel_exinput

graf_dragbox

Let the user move a box around the screen

Class: AES *Category: Graphics Handling*

SYNOPSIS

```
#include <aes.h>

res=graf_dragbox(w,h,sx,sy,bx,by,bw,bh,lastx,lasty);

int res;          error return
int w;            width of box
int h;            height of box
int sx;           initial x position
int sy;           initial y position
int bx;           x co-ordinate of bounding rectangle
int by;           y co-ordinate of bounding rectangle
int bw;           width of bounding rectangle
int bh;           height of bounding rectangle
short *lastx;     final x-coordinate of box
short *lasty;     final y-coordinate of box
```

DESCRIPTION

This function lets the user drag a box of a fixed size given by the w and h parameters. This box starts at (sx, sy) and the user will not be able to drag this outside the bounding rectangle given by (bx, by, bw, bh).

The final position of the box (i.e. when the user releases the left mouse button) is returned in the lastx and lasty parameters.

RETURNS

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

SEE

graf_rubberbox, graf_slidebox

graf_growbox

Draw a growing box

Class: AES *Category: Graphics Handling*

SYNOPSIS

```
#include <aes.h>

res=graf_growbox(x1,y1,w1,h1,x2,y2,w2,h2);

int res;        error return
int x1;         initial x co-ordinate of box
int y1;         initial y co-ordinate of box
int w1;         initial width of box
int h1;         initial height of box
int x2;         final x co-ordinate of box
int y2;         final y co-ordinate of box
int w2;         final width of box
int h2;         final height of box
```

DESCRIPTION

This function draws a box growing from a box with top left corner (x1, y1) with width w1 and height h1 to a box with top left corner (x2, y2) with width w2 and height h2. Note that the larger rectangle is second.

This call is usually used to provide a visual clue to the user. If the 'clue' does not pass any useful information to the user then the call should not be used.

RETURNS

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

SEE

graf_shrinkbox

graf_handle

Find the GEM VDI handle used by the AES

Class: AES *Category: Graphics Handling*

SYNOPSIS

```
#include <aes.h>

handle=graf_handle(wchar,hchar,wbox,hbox);

int handle;       VDI handle being used by the AES
short *wchar;     width of character cell in pixels
short *hchar;     height of character cell in pixels
short *wbox;      width of box surrounding a character
short *hbox;      height of box surrounding a character
```

DESCRIPTION

In addition to finding the GEM VDI handle being used by the AES, this function also returns the size of a character in the system font in pixels. This is the font that the AES uses when drawing normal text in object trees. The width and height (in pixels) of a box that surrounds a single character font is also returned; this is the minimum size of a G_BOXCHAR object.

Normally applications are not interested in this character size information, so they just pass an unused variable for each of the four parameters. See the example below.

RETURNS

The function returns the GEM VDI handle being used by the AES. The application can then open a virtual workstation using the VDI function v_opnvwk and then make further VDI calls to draw text and graphics on the screen.

SEE

v_opnvwk

EXAMPLE

```
#include <aes.h>

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

  appl_init();
  handle=graf_handle(&junk,&junk,&junk,&junk);
...
}
```

graf_mkstate

Return the current mouse status

Class: AES *Category: Graphics Handling*

SYNOPSIS

```
#include <aes.h>

res=graf_mkstate(x,y,button,kstate);

int res;              reserved: always 1 at present
short *x;             mouse x co-ordinate
short *y;             mouse y co-ordinate
short *button;        mouse button state
short *kstate;        keyboard shift state
```

DESCRIPTION

This function returns the current mouse position in (x, y) together with the current state of the mouse buttons in the button parameter. This parameter is a bitmap with bit 0 indicating the left mouse button and bit 1 the right mouse button. A bit is set if the appropriate mouse button is down. Thus if just the left button is down then 1 is returned in the button parameter.

The kstate parameter gives the state of the shift keys depressed; this is also a bitmap with the following meanings:

Name	Value	Meaning
K_RSHIFT	0x0001	Right shift key depressed
K_LSHIFT	0x0002	Left shift key depressed
K_CTRL	0x0004	Ctrl key depressed
K_ALT	0x0008	Alt key depressed

RETURNS

The function return value is reserved. This is always 1 at present.

graf_mouse

Change the mouse form

Class: AES *Category: Graphics Handling*

SYNOPSIS

```
#include <aes.h>

res=graf_mouse(number,formaddr);

int  res;               error return
int  number;            mouse form
void  *formaddr;        pointer to user defined form
```

DESCRIPTION

This function changes the appearance of the mouse according to the value of the number parameter:

Name	Value	Meaning
ARROW	0	Arrow.
TEXT_CRSR	1	Text cursor (vertical bar).
HOURGLASS	2	Busy bee.
POINT_HAND	3	Pointing finger.
FLAT_HAND	4	Extended fingers.
THIN_CROSS	5	Thin cross hair.
THICK_CROSS	6	Thick cross hair.
OUTLN_CROSS	7	Outline cross hair.
USER_DEF	255	User defined mouse form given by the buffer pointed to by formaddr. See below.
M_OFF	256	Hide mouse.
M_ON	257	Show mouse.

The structure pointed to by formaddr is the same as the MFORM structure defined in vdi.h and described under vsc_form.

The AES convention is that non-arrow cursors should only be used inside the work area of the current window. If your program is using another mouse form then it should use the mouse event facilities of evnt_multi to change the mouse form as the mouse enters and leaves the work area of your window.

The M_OFF and M_ON parameters are the most frequently used; so that your program can hide the mouse whilst writing to the display. These calls nest, so ensure that for every call on M_OFF, there is a call to M_ON otherwise the mouse will not reappear when M_ON is used.

Note that for calls other than USER_DEF the formaddr parameter is not required and the value NULL should be used.

RETURNS

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

SEE

vsc_form, v_show_c, v_hide_c

graf_movebox

Draw a moving box

Class: AES *Category: Graphics Handling*

SYNOPSIS

```
#include <aes.h>

res=graf_movebox(w,h,sx,sy,ex,ey);

int res;        error return
int w;          width of box
int h;          height of box
int sx;         initial x position
int sy;         initial y position
int ex;         final x position
int ey;         final y position
```

DESCRIPTION

This function draws a box of width w and height h moving from position (sx, sy) to (ex, ey). Naturally this is very fast on the ST.

RETURNS

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

SEE

graf_growbox, graf_shrinkbox

graf_rubberbox

Let the user drag a rubber box

Class: AES *Category: Graphics Handling*

SYNOPSIS

```
#include <aes.h>

res=graf_rubberbox(x,y,minw,minh,lastw,lasth);

int res;              error return
int x;                x co-ordinate of rectangle
int y;                y co-ordinate of rectangle
int minw;             minimum width of rectangle
int minh;             minimum height of rectangle
short *lastw;         final width of box
short *lasth;         final height of box
```

DESCRIPTION

This function lets the user drag a rubber box with top left hand corner starting at (x, y). The minimum size of the rectangle is passed in the minw and minh parameters.

The final width and height of the rectangle (i.e. when the user releases the left mouse button) are returned in the lastw and lasth parameters.

RETURNS

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

SEE

graf_dragbox

graf_shrinkbox — Draw a shrinking box

Class: AES — *Category: Graphics Handling*

SYNOPSIS

```
#include <aes.h>

res=graf_shrinkbox(x1,y1,w1,h1,x2,y2,w2,h2);

int res;        error return
int x1;         final x co-ordinate of box
int y1;         final y co-ordinate of box
int w1;         final width of box
int h1;         final height of box
int x2;         initial x co-ordinate of box
int y2;         initial y co-ordinate of box
int w2;         initial width of box
int h2;         initial height of box
```

DESCRIPTION

This function draws a box shrinking from a box with top left corner (x2, y2) with width w2 and height h2 to a box with top left corner (x1, y1) with width w1 and height h1.

Note that the larger and initial rectangle is second.

RETURNS

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

SEE

graf_growbox

graf_slidebox

Let the user slide a box within its parent

Class: AES *Category: Graphics Handling*

SYNOPSIS

```
#include <aes.h>

res=graf_slidebox(tree,parent,object,vertical);

int res;          position of object relative to parent
OBJECT *tree;     object tree
int parent;       parent of object to slide
int object;       object that is to be move
int vertical;     1 vertical movement, 0 for horizontal
```

DESCRIPTION

This function will let the user slide a given box (with index object in the form tree) within its parent (with index parent). If the movement is to be vertical then 1 should be passed in the vertical parameter, otherwise the value zero, indicating horizontal movements.

RETURNS

The function returns a value in the range 0 to 1000, giving the position of the object relative to the parent.

SEE

graf_dragbox, objc_draw

EXAMPLE

```
/*
 * demonstrate a slider bar using a builtin resource
 * Much easier done using WERCS!
 */

#include <aes.h>

OBJECT tree[] = {
  {-1,1,4,G_IBOX,0x0,0x0,(void *)0x1181,0,0,1026,13},
  {3,2,2,G_BOX,0x40,0x0,(void *)0x111c1,
    0,2049,1026,10},
  {1,-1,-1,G_BOX,0x40,0x0,(void *)0x11181,
    0,0,1026,2048},
  {4,-1,-1,G_BOXCHAR,0x40,0x0,(void *)0x1011181,
    0,0,1026,2049},
  {0,-1,-1,G_BOXCHAR,0x60,0x0,(void *)0x2011181,
    0,2059,1026,2049},
};
```

```
#define BAR 1
#define SLIDER 2

void do_slider(void)
{
  fix_tree(slider);   /* fix up co-ords in our tree */
  draw_tree(tree)     /* render the tree on-screen */
  /* give a slider effect */
  pos=graf_slidebox(tree,BAR,SLIDER,1);
  /* calculate the new object position */
  tree[SLIDER].ob_y=umul_div(pos,
    tree[BAR].ob_height-tree[SLIDER].ob_height,1000);
}
```

graf_watchbox

Track mouse relative to an object

Class: AES *Category: Graphics Handling*

SYNOPSIS

```
#include <aes.h>

res=graf_watchbox(tree,obj,instate,outstate);

int res;        1 if the mouse is in the box,
                0 if outside
OBJECT *tree;   object tree
int obj;        index of object to watch
int instate;    object state when mouse is inside box
int outstate;   object state when mouse is outside box
```

DESCRIPTION

This function will change the state of the given object as the mouse moves inside and outside of the box.

The object is specified by tree and obj (the object index) as usual and the value for the ob_state field when inside the box is passed in instate and that for outside the box in outstate.

This function should only be called when the mouse button is down and inside the box. graf_watchbox returns when the mouse is released.

RETURNS

The function returns 1 if the mouse is inside the box when the button is released and 0 if the mouse is outside the box.

SEE

graf_mkstate, graf_slidebox

menu_bar

Display or de-install the menu bar

Class: AES *Category: Menu Handling*

SYNOPSIS

```
#include <aes.h>

res=menu_bar(tree,show);

int res;            error return
OBJECT *tree;       object tree
int show;           1 means display bar,
                    0 means de-install
```

DESCRIPTION

This function informs the AES that it should use the object tree as its menu bar if the show parameter is 1. Object trees that are to be used as menu bars must conform to strict rules and as a result they are best designed with WERCS and then loaded from a resource file.

Once the menu has been installed the AES will send your program menu event messages when the items are selected, which can be detected using evnt_mesag and evnt_multl.

If you have used this function then you should call menu_bar with show set to 0 before exiting. Note however that this does not actually erase the bar from the screen.

RETURNS

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

SEE

rsrc_gaddr, evnt_mesag, evnt_multi

menu_icheck

Display/Erase a menu item check mark

Class: AES *Category: Menu Handling*

SYNOPSIS

```
#include <aes.h>

res=menu_icheck(tree,item,check);

int res;              error return
OBJECT *tree;         object tree
int item;             index of item to check
int check;            1 means display mark,
                      0 means don't
```

DESCRIPTION

This function can be used to display a check (or tick) mark by a menu item. The item index is normally obtained from the header file produced by WERCS.

Any check mark by an item can be cleared by calling this function with a check parameter of 0, or displayed by using a parameter of 1.

RETURNS

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

SEE

evnt_mesag, evnt_multi

menu_ienable

Enable/Disable a menu item

Class: AES *Category: Menu Handling*

SYNOPSIS

```
#include <aes.h>

res=menu_ienable(tree,item,enable);

int res;               error return
OBJECT *tree;          object tree
int item;              index of item to enable/disable
int enable;            1 means enable, 0 means disable
```

DESCRIPTION

This function can be used to dim (or disable) a menu item if the parameter enable is zero. The item index is normally obtained from the header file produced by WERCS.

If a menu item has been disabled and you wish to re-enable it then call this function with a enable parameter of 1, alternatively to disable an entry set the enable parameter to 0. Note also that on TOS version 1.2 and above it is also possible to disable menu titles (rather than just the items) using this call.

RETURNS

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

SEE

menu_bar, menu_icheck, menu_tnormal

menu_register

Register a desk accessory with the AES

Class: AES *Category: Menu Handling*

SYNOPSIS

```
#include <aes.h>

item=menu_register(ap_id,text);

int item;                   error return or item number
int ap_id;                  application identifier
const char *text;           the text to display
```

DESCRIPTION

This function is used to insert a menu entry for a desk accessory in the Desk menu. The text for the menu entry is passed as the text parameter and the application identifier (ap_id) is as returned from the appl_init call.

RETURNS

The function returns -1 if the entry cannot be added to the Desk menu or the positive menu item number if it has been added.

SEE

menu_bar, menu_icheck, menu_tnormal

EXAMPLE

See the example supplied on disk (chdiracc.c).

menu_text
Change the text of a menu item

Class: AES *Category: Menu Handling*

SYNOPSIS

```
#include <aes.h>

res=menu_text(tree,item,text);

int res;                    error return
OBJECT *tree;               object tree
int item;                   index of item to change
const char *text;           the text to display
```

DESCRIPTION

This function can be used to change the text of a given menu item. The item index is normally obtained from the header file produced by WERCS.

The new text should not be longer than the original length of the message.

RETURNS

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

menu_tnormal Display a menu title in normal/inverse video

Class: AES *Category: Menu Handling*

SYNOPSIS

```
#include <aes.h>

res=menu_tnormal(tree,item,normal);

int res;                    error return
OBJECT *tree;               object tree
int item;                   index of item to change
int normal;                 1 means normal,
                            0 means inverse
```

DESCRIPTION

This function can be used to show a menu item or title in inverse video if the parameter normal is zero. The item index is normally obtained from the header file produced by WERCS.

Calling this function with a normal parameter of 1, will restore an item to normal video. This is often used after a menu event has occurred because the AES will display the menu title in inverse video, so your program can use this function to return it to normal.

RETURNS

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

SEE

evnt_mesag, evnt_multi, menu_bar, menu_icheck, menu_ienable

EXAMPLE

```
/* dispatch menu events */
#include <aes.h>
void do_menu(OBJECT *menu)
{
  short msg[8];

  evnt_mesag(msg);
  if (msg[0]==MN_SELECTED)
  {
    switch (msg[4])
    {
      case ...
        break;
    }
    menu_tnormal(menu,msg[3],1);
  }
}
```

objc_add

Add an object to an object tree

Class: AES *Category: Object Manipulation*

SYNOPSIS

```
#include <aes.h>

res=objc_add(tree,parent,child);

int res;        error return status
OBJECT *tree;   tree in which the child is to be added
int parent;     the index of the object's parent
int child;      the index of the object to be added
```

DESCRIPTION

This function updates the ob_next, ob_head and ob_tail fields of the appropriate objects so that the object within the tree is added to the tree structure with the appropriate parent.

The ob_next, ob_head and ob_tail fields of the object being added should be initialised to NIL before calling this function. The other fields may be set up as required.

The object tree structure is described in detail in Volume I.

RETURNS

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

SEE

objc_delete

objc_change — Change and possibly display an object's state

Class: AES — *Category: Object Manipulation*

SYNOPSIS

```
#include <aes.h>

res=objc_change(tree,object,rsvd,x,y,w,h,state,draw);

int res;           error return status
OBJECT *tree;      object tree
int object;        the object to change
int rsvd;          reserved for future use
int x;             x co-ordinate of the clipping
                   rectangle
int y;             y co-ordinate of the clipping
                   rectangle
int w;             width of the clipping rectangle
int h;             height of the clipping rectangle
int state;         the new object state
int draw;          if 1 then re-draw object
                   if 0 don't
```

DESCRIPTION

This function changes the given object's ob_state field to be state. If the draw parameter is 1 then the object is re-drawn subject to the clipping rectangle given by the x, y, w and h parameters. These are screen co-ordinates. The reserved parameter rsvd *must* be given the value zero.

The object structure is described in detail in Volume I.

If the draw parameter is 0 then the object is not re-drawn. In this case it is generally clearer and quicker to manipulate the object tree directly.

RETURNS

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

SEE

objc_draw

objc_delete

Delete a n object from an object tree

Class: AES *Category: Object Manipulation*

SYNOPSIS

```
#include <aes.h>

res=objc_delete(tree,obj);

int res;           error return status
OBJECT *tree;      tree containing object to be deleted
int obj;           the index of the object
```

DESCRIPTION

This function updates the ob_next, ob_head and ob_tail fields of the appropriate objects so that the object obj is deleted from the tree structure.

This function will not move other objects in the tree structure. This function is the converse of objc_add.

The object tree structure is described in detail in Volume I.

RETURNS

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

SEE

objc_add

objc_draw

Draw part or all of an object tree

Class: AES *Category: Object Manipulation*

SYNOPSIS

```
#include <aes.h>

res=objc_draw(tree,startobj,depth,x,y,w,h);

int res;            error return
OBJECT *tree;       object tree to be drawn
int startobj;       index of the first object to draw
int depth;          the depth of objects to draw
int x;              x co-ordinate of the clipping
                    rectangle
int y;              y co-ordinate of the clipping
                    rectangle
int w;              width of the clipping rectangle
int h;              height of the clipping rectangle
```

DESCRIPTION

This function draws part or all of an object tree (normally a dialog box).

If the object tree is stored in a resource file then rsrc_gaddr is normally used to find the address of the tree.

The first object to draw is given by the startobj parameter; to draw the whole tree use the value ROOT.

If the depth parameter is zero then only the startob object will be drawn; if depth is 1 then this object and its first generation children will be displayed, etc. To draw all the children use the value MAX_DEPTH.

The x, y, w and h parameters give a clipping rectangle so that only part of the screen may be updated. Note that if your root object has a border or is outlined, then don't use its co-ordinates for the clipping rectangle, otherwise the border or outline may not all be drawn.

RETURNS

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

SEE

rsrc_gaddr, form_do

objc_edit

Form processing support routine

Class: AES *Category: Object Manipulation*

SYNOPSIS

```
#include <aes.h>

res=objc_edit(tree,object,ch,curpos,kind);

int res;            error return status
OBJECT *tree;       object tree
int object;         the current object
int ch;             key pressed by user
short *curpos;      cursor position in raw text
int kind;           action to perform
```

DESCRIPTION

This function is only normally used when writing your own form handler rather than using the standard form_do. The object must be an editable text field.

The action performed depends on the value of kind as follows:

ED_START	Reserved for future use. Do not call.
ED_INIT	Displays the text cursor for this object and returns in curpos the initial position of the cursor within the te_ptext field. This will be at the end of the string.
ED_CHAR	This is used to validate the input character ch against the template, updating the te_ptext field and curpos as appropriate. curpos must be set up correctly before this call. After such a call curpos will be updated so that it may be used for another ED_CHAR call.
ED_END	Turns off the text cursor.

RETURNS

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

SEE

form_keybd, form_button

objc_find

Find which object is 'under' a given co-ordinate

Class: AES *Category: Object Manipulation*

SYNOPSIS

```
#include <aes.h>

res=objc_find(tree,startobj,depth,x,y);

OBJECT *tree;    object tree to be searched
int startobj;    index of first object to consider
int depth;       the depth of objects to search
int x;           x co-ordinate of the point to find
int y;           y co-ordinate of the point to find
```

DESCRIPTION

This function searches all or part of a tree to find which object lies 'under' a given co-ordinate. It is often used to find which item the user has selected by clicking with the mouse.

The first object to consider is given by the startobj parameter; to search the whole tree use the value ROOT.

If the depth parameter is zero then only the startob object will be considered; if depth is 1 then this object and its first generation children will be searched etc. To search to the maximum depth of children use the value MAX_DEPTH.

The x and y parameters give the point to search for in screen co-ordinates.

RETURNS

The function returns the object index of the object that was found or -1 if the object was not found.

SEE

objc_draw

EXAMPLE

See form_button for an example of objc_find.

objc_offset

Find object's screen co-ordinates

Class: AES *Category: Object Manipulation*

SYNOPSIS

```
#include <aes.h>

error=objc_offset(tree,object,x,y);

int error;          error code
OBJECT *tree;       object tree
int object;         index of object within tree
short *x;           x co-ordinate relative to screen
short *y;           y co-ordinate relative to screen
```

DESCRIPTION

This function returns in (x,y) the screen co-ordinates of object from the given tree. Remember that internally an object's co-ordinates are represented as offsets from its parent.

RETURNS

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

SEE

objc_xywh

objc_order

Move an object within its list of siblings

Class: AES *Category: Object Manipulation*

SYNOPSIS

```
#include <aes.h>

res=objc_order(tree,object,action);

OBJECT *tree;     object tree containing the
                  structure
int object;       the object to move
int action;       where to move the object
```

DESCRIPTION

This function updates the ob_next, ob_head and ob_tail fields of the appropriate objects so that the tree is re-ordered relative to its siblings. Thus, for example, you may change an object from being the second child of its parent to being the first child.

The possible values for the action parameter are as follows:

-1	make the object the last child
0	make the object the first child
1	make the object the second child
....	

RETURNS

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

SEE

objc_draw

objc_walk

Iteratively walk an object tree

Class: Lattice *Category: Object Manipulation*

SYNOPSIS

```
#include <aes.h>

objc_walk(tree,first,last,reject,routine);

OBJECT *tree;                        object tree
int first;                           starting object
int last;                            final object
int reject;                          flags to ignore
int (*routine)(tree,object);         user routine
int object                           object found
```

DESCRIPTION

This function can be used to 'walk' an object tree (i.e. call a routine for each object) without writing code that explicitly accesses each of the ob_tail, ob_head and ob_next fields.

first gives the index in the tree to start walking at. This should be ROOT to walk the entire tree.

The walk will stop when the index stop is reached without calling the routine for this object. To search the entire tree use a value of NIL.

reject will normally be HIDETREE to ignore any hidden parts of the tree as the reject parameter is 'ANDed' with the ob_flags field of the next object being considered and if this is non-zero then this object and any of its children are ignored. Thus, using a value of 0 for reject will cause the entire tree including any hidden parts to be scanned. You could also use this parameter to ignore objects that are radio buttons!

routine gives the function to be called for each object that satisfies the criteria above. It takes two parameters; the first is the object tree and the second is the current object number. This function should return 0 if any sub-trees of this object are to be searched and 1 if any children are to be ignored.

Note that this function is an extension to the standard bindings and so will be non-portable to other C implementations.

EXAMPLE

```
/*
 * this example un-hides every element in a tree
 */

#include <aes.h>

/*
 * unhides the object cur in the given tree
 */
int unhide(OBJECT *tree,int cur)
{
  /*
   * clear the appropriate bit of the ob_flags field
   */
  tree[cur].ob_flags&=~HIDETREE;
  return 0; /* means continue */
}

/*
 * perform unhide for the whole tree starting at ROOT
 * looking at all branches including hidden ones
 */
objc_walk(tree,ROOT,NIL,0,unhide);
```

objc_xywh — Find object's screen co-ordinates as a rectangle

Class: Lattice *Category: Object Manipulation*

SYNOPSIS

```
#include <aes.h>

error=objc_xywh(tree,object,rect);

int error;          error code
OBJECT *tree;       object tree
int object;         index of object within tree
GRECT *rect;        a pointer to the co-ordinates
```

DESCRIPTION

This function returns in (rect.g_x, rect.g_y) the screen co-ordinates of object from the given tree together with its width and height in rect.g_w and rect.g_h.

If you are using the GRECT structure rather than individual x, y, width and height co-ordinates then we recommend that you use this function rather than objc_offset. Be aware, however, that this is an extension to the standard bindings.

RETURNS

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

SEE

objc_offset, rc_equal

rc_constrain — Constrain one rectangle within another

Class: Lattice *Category: Rectangle Handling*

SYNOPSIS

```
#include <aes.h>

rc_constrain(rect1,rect2);

const GRECT *rect1;       one rectangle to use
GRECT *rect2;             the target rectangle
```

DESCRIPTION

This function can be used to ensure that rect2 lies within rect1. The co-ordinates of rect2 will be updated so that this is the case.

SEE

rc_equal

EXAMPLE

```
/*
 * force a window to remain inside the desktop
 * after a move request
 */

#include <aes.h>

void do_move(int wh,GRECT *p)
{
  GRECT q;

  /* find size of desktop window */
  wind_get(DESK,WF_CXYWH,&q.g_x,&q.g_y,&q.g_w,&q.g_h);
  rc_constrain(&q,p);
  /* actually move the window */
  wind_set(wh,WF_CXYWH,p->g_x,p->g_y,p->g_w,p->g_h);
}
```

rc_copy

Copy one rectangle to another

Class: Lattice *Category: Rectangle Handling*

SYNOPSIS

```
#include <aes.h>

rc_copy(source,dest);

const GRECT *source;     the source rectangle
GRECT *dest;             the destination rectangle
```

DESCRIPTION

This function copies the rectangle source to the rectangle dest. This function is only provided for compatibility with older compilers; a structure assignment is much clearer.

SEE

rc_equal

EXAMPLE

```
#include <aes.h>

int main(void)
{
  GRECT r1,r2;

  rc_copy(&r1,&r2);
  /* is the same as */
  r2=r1;
}
```

rc_equal

Compare one rectangle with another

Class: Lattice *Category: Rectangle Handling*

SYNOPSIS

```
#include <aes.h>

equal=rc_equal(rect1,rect2);

int equal;              zero if the rectangles differ
const GRECT *rect1;     the first rectangle to compare
const GRECT *rect2;     the second rectangle to compare
```

DESCRIPTION

This function compares whether two rectangles are equal. The GRECT structure is a generally useful one for manipulating AES rectangles, although it is not part of the standard GEM bindings. It is defined, in aes.h, as:

```
typedef struct grect
{
  short g_x;          x co-ordinate
  short g_y;          y co-ordinate
  short g_w;          width of rectangle
  short g_h;          height of rectangle
} GRECT;
```

You can use this just like one of your own C structures if you need to access the individual fields yourself.

RETURNS

This function returns 1 if the two rectangles are equal and 0 otherwise.

EXAMPLE

```
#include <aes.h>

int main(void)
{
  GRECT r1,r2;

  r1=r2;
  if (rc_equal(&r1,&r2))
    printf("this code would get executed\n");
  return 0;
}
```

rc_inside

Test whether a point is within a rectangle

Class: Lattice *Category: Rectangle Handling*

SYNOPSIS

```
#include <aes.h>

res=rc_inside(x,y,rect);

int res                     0 if point is outside rect
int x;                      x co-ordinate to test
int y;                      y co-ordinate to test
const GRECT *rect;          rectangle to use
```

DESCRIPTION

This function tests whether a point (x, y) is within the given rectangle.

RETURNS

This function returns 1 if the point is inside the rectangle and 0 if it is outside.

SEE

rc_equal

rc_intersect

Find the intersection of two rectangles

Class: Lattice *Category: Rectangle Handling*

SYNOPSIS

```
#include <aes.h>

res=rc_intersect(rect1,rect2);

int res;                 1 if intersection is non-empty
const GRECT *rect1;      the first rectangle
GRECT *rect2;            the target rectangle
```

DESCRIPTION

This function finds the intersection of two rectangles, if any. The resulting rectangle is placed in rect2. rect2 will be modified even if there is no intersection. This can be used when re-drawing windows; it is used by wind_redraw, for example.

RETURNS

This function returns 1 if the intersection is non-empty, or 0 if there is no intersection.

SEE

rc_equal, wind_redraw

EXAMPLE

```
/*
 * Implement wind_redraw, a window redraw primitive
 */

#include <aes.h>

int wind_redraw (int w_hand,GRECT *area,
  int (*redraw)(int,GRECT *))
{
  GRECT box;
  int ok=1;

  graf_mouse(M_OFF,NULL); /* hide the mouse */
  /* suppress menu drops */
  wind_update(BEG_UPDATE);
  /* get the first rectangle on the windows list */
  wind_get(w_hand,WF_FIRSTXYWH,&box.g_x,&box.g_y,
    &box.g_w,&box.g_h);
```

```
  /* while the box exists */
  while (box.g_w && box.g_h)
  {
    /* find the intersection with the redraw area */
    if (rc_intersect(area,&box))
      /* call the users redraw routine */
      if (!(ok=redraw(w_hand,&box)))
        break;
    /* fetch the next rectangle on the windows list */
    wind_get(w_hand,WF_NEXTXYWH,&box.g_x,&box.g_y,
      &box.g_w,&box.g_h);
  }
  /* release the menu suspension */
  wind_update(END_UPDATE);
  /* and re-plot the mouse */
  graf_mouse(M_ON,NULL);
  return ok;
}
```

rc_union

Find the union of two rectangles

Class: Lattice *Category: Rectangle Handling*

SYNOPSIS

```
#include <aes.h>

rc_union(rect1,rect2);

const GRECT *rect1;           the first rectangle
GRECT *rect2;                 the target rectangle
```

DESCRIPTION

This function finds the union of two rectangles, i.e. the smallest rectangle that contains both rect1 and rect2. The resulting rectangle is placed in rect2.

SEE

rc_equal

rsrc_free

Free memory used by a resource file

Class: AES *Category: Resource File Handling*

SYNOPSIS

```
#include <aes.h>

res=rsrc_free(void);

int res;    error return;
```

DESCRIPTION

This function frees the memory allocated by rsrc_load. If your application needs its resource file until it terminates then there is no need to call this function as the memory will be freed on termination.

RETURNS

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

SEE

rsrc_load

rsrc_gaddr — Get the address of a resource file data item

Class: AES — *Category: Resource File Handling*

SYNOPSIS

```
#include <aes.h>

res=rsrc_gaddr(type,index,addr);

int res;        error return;
int type;       type of item to look for
int index;      number of item to look for
void *addr;     where to store the address of the item
```

DESCRIPTION

This function is used find the address of an item that has been loaded using rsrc_load. The types of items that can be looked for are as follows:

R_TREE	object tree
R_OBJECT	individual object
R_TEDINFO	TEDINFO field
R_ICONBLK	ICONBLK field
R_BITBLK	BITBLK field
R_STRING	string
R_IMAGEDATA	image data
R_OBSPEC	ob_spec within the objects
R_TEPTEXT	te_ptext within the tedinfos
R_TEPTMPLT	te_ptmplt within the tedinfos
R_TEPVALID	te_pvalid within the tedinfos
R_IBPMASK	ib_pmask within the iconblks
R_IBPDATA	ib_pdata within the iconblks
R_IBPTEXT	ib_ptext within the iconblks
R_BIPDATA	bi_pdata within the bitblks
R_FRSTR	pointer to a free string
R_FRIMG	pointer to a free image

The index parameter is the index of this particular sort of item in the file. The address found by rsrc_gaddr is returned by storing it at the address given by addr.

Most of the item types are not of much use because WERCS, and all the other resource construction sets that we know of, only return the indices within files of trees, free strings and free images. Thus the R_TREE, R_FRSTR and R_FRIMG parameters are all useful.

WERCS also provides the object indices for objects within each individual tree. This is not the same as the value that rsrc_gaddr wants; that is the offset within the entire resource file. These are actually the same for the first tree in the file, but there is little point in taking advantage of this as your code won't work for subsequent trees.

The usual method to find the address of an object is to find the address of the tree using rsrc_gaddr(R_TREE, ...) and then treat the returned value as an array of objects. To find, say, the address of a te_ptext field within such an object, you follow the object tree data structure. This is described in more detail in Volume I.

RETURNS

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

SEE

rsrc_load

rsrc_load

Load a resource file

Class: AES — *Category: Resource File Handling*

SYNOPSIS

```
#include <aes.h>

res=rsrc_load(fname);

int res;                     error return;
const char *fname;           file name load
```

DESCRIPTION

This function is used to load resource files into memory and is passed a standard string. The resource file will be loaded into GEMDOS free memory and the co-ordinates within it are updated for the current screen resolution. The address of the items within the file can then be found using the rsrc_gaddr function.

Resource files are normally created using WERCS.

RETURNS

The function returns 0 if an error occurred (such as the file doesn't exist or there is insufficient memory) or non-zero otherwise.

SEE

rsrc_gaddr, rsrc_free

EXAMPLE

```
/*
 * load myrsc.rsc from disk
 */

#include  aes.h>

int get_rsc(void)
{
  int ok;

  ok=rsrc_load("MYRSC.RSC");
  if (!ok)
    form_alert(1,"[3][Can't load resource file][OK]");
  return ok;
}
```

rsrc_obfix

Convert a n object to screen co-ordinates

Class: AES *Category: Resource File Handling*

SYNOPSIS

```
#include <aes.h>

res=rsrc_obfix(tree,index);

int res;            reserved
OBJECT *tree;       type of item to look for
int index;          index of the object to change
```

DESCRIPTION

This function can be used to convert an object's co-ordinates from character co-ordinates (where the low byte specifies the number of characters and the high byte the pixel offset within this) to screen pixel co-ordinates (as required by objc_draw). Character co-ordinates (with pixel deltas) are used in resource files. The rsrc_obfix call is used by rsrc_load and can be used to fix up your own embedded resources or custom resource files. The tree parameter gives the tree to use and the object parameter the index of the desired object within that tree. Note that this function fixes only a *single* object and not a complete tree.

Also beware that rsrc_obfix has some special cases, in particular it will increase/decrease the width of 80 character wide menus for different sized screens.

RETURNS

The function result is reserved. At present 1 is always returned.

SEE

rsrc_load, objc_draw

EXAMPLE

```
/*
 * routine to fix an entire object tree
 */
#include <aes.h>

void fix_tree(OBJECT *tree)
{
  /* walk a tree until we find the last object */

  while (!(tree->ob_flags&LASTOB))
    rsrc_obfix(tree++,0);
}
```

rsrc_saddr

Set the address of a resource file data item

Class: AES *Category: Resource File Handling*

SYNOPSIS

```
#include  <aes.h>

res=rsrc_saddr(type,index,addr);

int  res;       error  return;
int  type;      type of  item  to  change
int  index;     number  of  item  to  change
void  *addr;    address  of  the  item  to  store
```

DESCRIPTION

This function is used to set the address of a free string or image item that has been loaded using rsrc_load.

The types of items that can be looked for are as follows:

R_FRSTR	Pointer to free string.
R_FRIMG	Pointer to free image.

The index parameter is the index of this particular sort of item in the file, i.e. that returned by WERCS.

This function may be used if you wish to move (for instance) a free string representing an alert to a new location.

RETURNS

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

SEE

rsrc_load, rsrc_gaddr

EXAMPLE

```
/*
 * set up a shared alert index
 */

#include <aes.h>

#define ALERT 0   /* constant from WERCS */

static buffer[100];

void setup_alert(const char *s)
{
  sprintf(buffer,"[2][%s][OK]",s);
  rsrc_saddr(R_FRSTR, ALERT, buffer);
  /*
   * rsrc_gaddr(R_FRSTR, ALERT, ... ) will now
   * return buffer
   */
}
```

scrp_read

Find name of the scrap directory

Class: AES *Category: Scrap Handling*

SYNOPSIS

```
#include <aes.h>

res=scrp_read(dirname);

int res;              error return
char *dirname;        current scrap directory name
```

DESCRIPTION

This function returns the name of the current scrap directory. If your program wants to read a disk based clipboard that has been set up by another application then this call can be used to find the directory where the clipboard file(s) are stored. Unfortunately there is no agreed convention on the format that this data should take, only that the name is always SCRAP, with the extension indicating the form of the data. The length of the array specified by dirname should be at least FMSIZE characters long. FMSIZE is defined in dos.h.

RETURNS

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

SEE

scrp_write

scrp_write

Change the name of the scrap directory

Class: AES *Category: Scrap Handling*

SYNOPSIS

```
#include <aes.h>

res=scrp_write(dirname);

int res;                error return
const char *dirname;    new scrap directory name
```

DESCRIPTION

This function changes the name of the current scrap directory. If your program wants to change the directory where it is storing a disk based clipboard that can be read by other applications then it should use this call. Unfortunately there is no agreed convention on the format that this data should take, only that the name is always SCRAP, with the extension indicating the form of the data.

RETURNS

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

SEE

scrp_read

shel_envrn

Search the AES's environment

Class: AES *Category: Shell Handling*

SYNOPSIS

```
#include <aes.h>

res=shel_envrn(value,name);

int res;                    reserved
char *value;                value returned
const char *name;           the environment variable
```

DESCRIPTION

This function can be used to search the AES's environment space for a particular environment variable. Initially this just contains PATH=, but unfortunately there is no way to add variables to this environment. If you are interested in the AES path then it is simpler to use shel_find to locate files.

The name parameter gives the variable name to search for *including* the equals (=) sign. value returns containing a pointer to the byte after the equals sign.

The getenv, putenv, rmvenv functions, from the main library, can be used to manipulate the GEMDOS environment.

RETURNS

The return value is reserved; the function returns 1 as present.

SEE

getenv, putenv, rmvenv, shel_find

shel_find

Find a file on the AES's search path

Class: AES *Category: Shell Handling*

SYNOPSIS

```
#include <aes.h>

res=shel_find(name);

int res;              error return
char *name;           the file name of the command
```

DESCRIPTION

This function can be used to find a file either in the current directory or on the AES's path. The file to search for is passed in name and the full pathname needed to access it is returned in the same parameter. As such this should be at least FMSIZE characters long, which is defined in the header file dos.h.

The AES's path is not the same as the GEMDOS path; it is the path that is used by rsrc_load and normally consists of just A:\ on floppy-based systems, or C:\ on hard disk systems. It may be changed however using the Saved! desk accessory. If your program requires files additional to a resource file, it should use shel_find to attempt to find them.

RETURNS

The function returns 0 if the file requested could not be located, or non-zero otherwise.

SEE

rsrc_load

EXAMPLE

```
/* find my .INF file */

#include <aes.h>
#include <dos.h>
#include <string.h>

char *get_inf(const char *s)
{
  static char buffer[FMSIZE];

  strcpy(buffer,s);
  strcat(buffer,".INF");
  if (shel_find(buffer))
    return buffer;
  return NULL;
}
```

shel_get

Read the AES's internal shell buffer

Class: AES *Category: Shell Handling*

SYNOPSIS

```
#include <aes.h>

res=shel_get(buff,len);

int res;                    error return
char *buff;                 buffer
int len;                    length to read
```

DESCRIPTION

This function reads the AES's internal shell buffer (the RAM version of the DESKTOP.INF file) into the buffer at the given address; len bytes will be read. The buffer should be at least 4192 bytes long to accomodate for TOS's later than AES version 1.40 (Rainbow TOS).

The corresponding function to write to this buffer is shel_put.

RETURNS

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

SEE

shel_put

EXAMPLE

See the example supplied on disk (rocp.c).

shel_put

Write to the AES's internal shell buffer

Class: AES *Category: Shell Handling*

SYNOPSIS

```
#include <aes.h>

res=shel_put(buff,len);

int res;                    error return
const char *buff;           buffer
int len;                    length to store
```

DESCRIPTION

This function writes into the AES's internal shell buffer (the RAM version of the DESKTOP.INF file) from the buffer at the given address. len bytes will be written. The length must not be greater than 1024 bytes for AES versions prior to 1.40 (Rainbow TOS) or 4192 bytes for later TOS's. If you write a new buffer to the AES, you must place a single ^Z (26 decimal) to indicate the end of the buffer.

The corresponding function to read this buffer is shel_get.

RETURNS

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

SEE

shel_get

shel_read

Find the command that invoked this program

Class: AES *Category: Shell Handling*

SYNOPSIS

```
#include <aes.h>

res=shel_read(name,tail);

int res;            reserved
char *name;         the name of the command
char *tail;         the command tail for this command
```

DESCRIPTION

This function can be used to find out the command that invoked this program and the program's command line, if the program was invoked by the desktop. It does not work if the program was run 'inside' another program.

A much better way to find the program's command line is to use the standard C argv and argc facilities, as described under the main function in Volume II.

RETURNS

The function returns 0 if an error occurred or non-zero otherwise. Note that the command tail returned has the same format as the GEMDOS Pexec command tail i.e. the first byte gives the length of the string.

SEE

main

shel_write

Run another application

Class: AES *Category: Shell Handling*

SYNOPSIS

```
#include <aes.h>

res=shel_write(ex,gr,over,name,tail);

int res;                 error return
int ex;                  normally 1
int gr;                  1 for GEM applications,
                         0 for TOS
int over;                1 to run afterwards
const char *name;        the file name of the command
const char *tail;        the command tail
```

DESCRIPTION

This function can be used to run another program when this application has finished. The ex parameter should be 1 to run another program. In theory this parameter can be 0 indicating that the Desktop should terminate when control returns to it, however this does not work on all current versions of the operating system.

The gr parameter specifies whether the program to be run is a .TOS (or .TTP) program (use 0 for this parameter) or a GEM (i.e. .PRG or .APP) program.

The name parameter specifies the complete filename (including extension) of the program to be run. The tail parameter specifies the command tail to be used in GEMDOS Pexec format i.e. the first byte gives the length of the string.

The over parameter should be 1 to run the program when control returns to the Desktop. Theoretically shel_write can be used to run other programs from within each other (with over=0), but this does not work due to a bug in all current versions of the operating system. To run a program inside the current one, you should use one of the fork family of functions. See Volume II details.

RETURNS

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

SEE

fork

EXAMPLE

```
/*
 * setup an application for running by the desktop
 */

#include <aes.h>

void setup_run(const char *cmd, const char *tail)
{
  char buf[128];

  strcpy(buf+1,tail);
  buf[0]=strlen(tail);
  shel_write(1,1,1,cmd,buf);
}
```

wind_calc	Work area to full size window co-ordinate mapping

Class: AES *Category: Window Handling*

SYNOPSIS

```
#include  <aes.h>

res=wind_calc(request,kind,x1,y1,w1,h1,
                                     x2,y2,w2,h2);

int  res;          error  return
int  request;      information  to  find
int  kind;         window  components  required
int  x1;           input  x  co-ordinate
int  y1;           input  y  co-ordinate
int  w1;           input  width
int  h1;           input  height
short  *x2;        output  x  co-ordinate
short  *y2;        output  y  co-ordinate
short  *w2;        output  width
short  *h2;        output  height
```

DESCRIPTION

This function returns the work area of a window with given components and border co-ordinates if the request parameter is WC_WORK or the border area of a window given the work area if the request parameter is WC_BORDER.

The components are specified using the kind parameter as for wind_create and are as follows:

NAME	Title bar with name.
CLOSE	Close box.
FULL	Full box.
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.

RETURNS

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

SEE

wind_create

EXAMPLE

```
/*
 * find the maximum work area of a fully configured
 * window
 */

#include <aes.h>

GRECT *get_max(void)
{
  static GRECT p;

  wind_get(DESK,WF_CXYWH,&p.g_x,&p.g_y,&p.g_w,&p.g_h);
  wind_calc(WC_WORK,NAME|CLOSE|FULL|MOVE|INFO|SIZE|
      UPARROW|DNARROW|VSLIDE|LFARROW|RTARROW|HSLIDE,
      p.g_x,p.g_y,p.g_w,p.g_h,
      &p.g_x,&p.g_y,&p.g_w,&p.g_h);
  return &p;
}
```

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

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 WF_WXYWH	The current work area of the window is returned.
WF_CURRXYWH WF_CXYWH	The current position and size of the window including borders.
WF_PREVXYWH WF_PXYWH	The co-ordinates of the previous window size including borders.
WF_FULLXYWH 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 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. 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

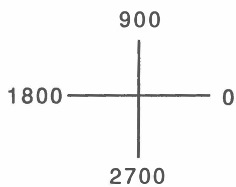
Class: VDI *Category: GDP Output*

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_allgn 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

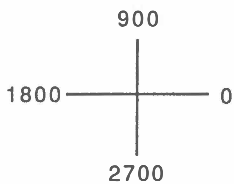
Class: VDI *Category: GDP Output*

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: 0 = Use NDC (normalised device co-ordinates) i.e. each page has co-ordinates 0 to 32767 regardless of the physical screen size. 1 = Reserved. 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: 0 = capable of precisely scaled image. 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: 0 = continuous scaling.
work_out(6)	Number of line types.

work_out(7)	Number of line widths: 0 = continuous scaling
work_out(8)	Number of marker types.
work_out(9)	Number of marker sizes: 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) to 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: 1 Bar. 2 Arc. 3 Pie slice. 4 Circle. 5 Ellipse. 6 Elliptical arc. 7 Elliptical pie. 8 Rounded rectangle. 9 Filled rounded rectangle. 10 Justified graphics text.
work_out(25) to work_out(34)	List of the attribute set used with each GDP: 0 Polyline. 1 Polymarker. 2 Text. 3 Fill area. 4 None.
work_out(35)	Colour capability flag: 0 no. 1 yes.

work_out(36)	Text rotation capability flag: 0 no. 1 yes.
work_out(37)	Fill area capability flag: 0 no. 1 yes.
work_out(38)	Cell array operation capability flag: 0 no. 1 yes.
work_out(39)	Number of available colours in palette: 0 continuous device (>32767 colours). 2 monochrome . >2 number of colours.
work_out(40)	Number of locator devices: 1 Keyboard only. 2 Keyboard and other input.
work_out(41)	Number of valuator devices: 1 Keyboard only. 2 Other valuator device is available.
work_out(42)	Number of choice devices: 1 function keys on keyboard. 2 if another keypad is available.
work_out(43)	Number of string devices: 1 keyboard.
work_out(44)	Workstation type: 0 output only. 1 input only. 2 input/output. 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();
}
```

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 saveaO,savea1;

  saveaO=getreg(REG_AO);
  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_AO,saveaO);

  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_plline 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_wldth).

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	LWHITE	Grey
BLACK	Black	LBLACK	Dark grey
RED	Red	LRED	Light blue
GREEN	Green	LGREEN	Blue green
BLUE	Blue	LBLUE	Light purple
CYAN	Dark blue	LCYAN	Dark purple
YELLOW	Brown	LYELLOW	Dark yellow
MAGENTA	Dark green	LMAGENTA	Light yellow

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

RETURNS

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

SEE

vs_color

EXAMPLE

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

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

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

vsf_interior, vsf_style	Set the fill style

Class: VDI *Category: Fill Area Attributes*

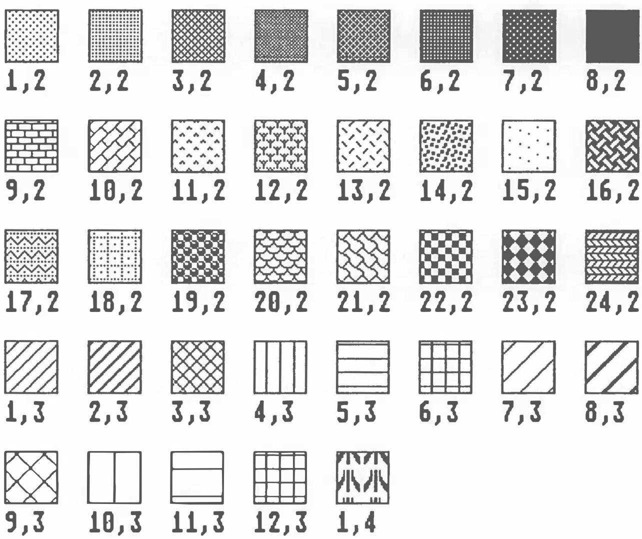
SYNOPSIS

```
#include  <vdi.h>

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

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

DESCRIPTION

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

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



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

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

RETURNS

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

SEE

vsf_udpat, vsf_perimeter, v_fillarea

vsf_perimeter

Set the fill area perimeter visibility

Class: VDI *Category: Fill Area Attributes*

SYNOPSIS

```
#include <vdi.h>

new_flag=vsf_perimeter(handle,flag);

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

DESCRIPTION

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

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

If flag is 0 then such borders are not drawn.

RETURNS

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

SEE

vsf_color, vqf_attributes, v_fillarea

vsf_udpat

Set the user defined fill pattern

Class: VDI *Category: Fill Area Attributes*

SYNOPSIS

```
#include <vdi.h>

vsf_udpat(handle,pattern,planes);

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

DESCRIPTION

This function changes the user defined fill pattern set using:

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

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

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

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

SEE

vsf_interior, vswr_mode

vsin_mode

Set input mode

Class: VDI *Category: Input Functions*

SYNOPSIS

```
#include  <vdi.h>

new_mode=vsin_mode(handle,dev_type,mode);

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

DESCRIPTION

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

The dev_type parameter should be one of

1	locator
2	valuator
3	choice
4	string

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

RETURNS

This function returns the new mode set.

SEE

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

vsl_color

Set the line colour

Class: VDI *Category: Line Attributes*

SYNOPSIS

```
#include <vdi.h>

new_col=vsl_color(handle,colour);

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

DESCRIPTION

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

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

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

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

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

RETURNS

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

SEE

vs_color

EXAMPLE

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

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

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

vsl_ends

Set the line end style

Class: VDI *Category: Line Attributes*

SYNOPSIS

```
#include <vdi.h>

new_col=vsl_ends(handle,begin,end);

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

DESCRIPTION

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

SQUARE	
ARROWED	
ROUND	

SEE

vsl_type, vsl_color

EXAMPLE

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

vsl_type, vsl_udsty

Set the line type

Class: VDI *Category: Line Attributes*

SYNOPSIS

```
#include <vdi.h>

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

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

DESCRIPTION

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

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

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

RETURNS

The vsl_type function returns the line type set.

SEE

vq_extnd

vsl_width

Set the line width

Class: VDI

Category: Line Attributes

SYNOPSIS

```
#include  <vdi.h>

new_size=vsl_width(handle,size);

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

DESCRIPTION

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

RETURNS

This function returns the line width actually set.

SEE

vq_extnd

EXAMPLE

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

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

vsm_choice

Choice input in sample mode

Class: VDI *Category: Input Functions*

SYNOPSIS

```
#include <vdi.h>

status=vsm_choice(handle,xout);

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

DESCRIPTION

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

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

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

RETURNS

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

SEE

vrq_choice, vsin_mode

vsm_color

Set the marker colour

Class: VDI *Category: Marker Attributes*

SYNOPSIS

```
#include <vdi.h>

new_col=vsm_color(handle,colour);

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

DESCRIPTION

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

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

WHITE 0	White	LWHITE	Grey
BLACK f	Black	LBLACK	Dark grey
RED 1	Red	LRED	Light blue
GREEN 2	Green	LGREEN	Blue green
BLUE 4	Blue	LBLUE	Light purple
CYAN	Dark blue	LCYAN	Dark purple
YELLOW 3	Brown	LYELLOW	Dark yellow
MAGENTA	Dark green	LMAGENTA	Light yellow

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

RETURNS

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

SEE

vs_color, vsm_type, vsm_height, vqm_attributes, v_pmarker

EXAMPLE

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

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

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

vsm_height

Set the marker height

Class: VDI *Category: Marker Attributes*

SYNOPSIS

```
#include <vdi.h>

new_size=vsm_height(handle,size);

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

DESCRIPTION

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

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

RETURNS

This function returns the marker height actually set.

SEE

vsm_color, vqm_attributes, vsm_type, v_pmarker

EXAMPLE

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

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

vsm_locator

Locator input in sample mode

Class: VDI *Category: Input Functions*

SYNOPSIS

```
#include <vdi.h>

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

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

DESCRIPTION

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

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

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

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

RETURNS

This function returns the following:

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

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

SEE

vrq_locator, vsin_mode

EXAMPLE

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

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

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

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

vsm_string

String input in sample mode

Class: VDI *Category: Input Functions*

SYNOPSIS

```
#include <vdi.h>

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

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

DESCRIPTION

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

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

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

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

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

RETURNS

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

SEE

vrq_string, vsin_mode

EXAMPLE

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

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

  char str[7];

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

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

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

vsm_type

Set the marker type

Class: VDI — *Category: Marker Attributes*

SYNOPSIS

```
#include <vdi.h>

new_type=vsm_type(handle,type);

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

DESCRIPTION

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

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

RETURNS

The function returns the marker type set.

SEE

vsm_color, vsm_height, v_pmarker

vsm_valuator

Valuator input in sample mode

Class: VDI *Category: Input Functions*

SYNOPSIS

```
#include <vdi.h>

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

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

DESCRIPTION

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

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

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

The status return values are as follows

0	Nothing happened
1	Valuator changed
2	Key press occurred

SEE

vrq_valuator, vsin_mode

vsp_message

Suppress palette messages

Class: VDI *Category: Palette Escape Functions*

SYNOPSIS

```
#include <vdi.h>

vsp_message(handle);

int handle;         workstation handle
```

DESCRIPTION

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

vsp_save

Save palette driver state

Class: VDI *Category: Palette Escape Functions*

SYNOPSIS

```
#include <vdi.h>

vsp_save(handle);

int handle;        workstation handle
```

DESCRIPTION

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

vsp_state

Set palette driver state

Class: VDI *Category: Palette Escape Functions*

SYNOPSIS

```
#include <vdi.h>

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

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

DESCRIPTION

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

vst_alignment

Set the base line for graphics text

Class: VDI *Category: Text Attributes*

SYNOPSIS

```
#include <vdi.h>

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

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

DESCRIPTION

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

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

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

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

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

SEE

v_gtext, v_justified

vst_color

Set the graphics text colour

Class: VDI *Category: Marker Attributes*

SYNOPSIS

```
#include <vdi.h>

new_col=vst_color(handle,colour);

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

DESCRIPTION

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

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

WHITE	White	LWHITE	Grey
BLACK	Black	LBLACK	Dark grey
RED	Red	LRED	Light blue
GREEN	Green	LGREEN	Blue green
BLUE	Blue	LBLUE	Light purple
CYAN	Dark blue	LCYAN	Dark purple
YELLOW	Brown	LYELLOW	Dark yellow
MAGENTA	Dark green	LMAGENTA	Light yellow

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

RETURNS

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

SEE

vs_color, v_gtext, vqt_attributes, v_justified

EXAMPLE

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

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

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

vst_effects

Set the graphics text effects

Class: VDI *Category: Text Attributes*

SYNOPSIS

```
#include <vdi.h>

new_effects=vst_effects(handle,effects);

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

DESCRIPTION

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

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

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

RETURNS

This function returns the text effects set.

SEE

v_gtext, v_justified, linea8

vst_font

Select particular GDOS font

Class: VDI *Category: Text Attributes*

SYNOPSIS

```
#include <vdi.h>

set_font=vst_font(handle,font);

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

DESCRIPTION

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

RETURNS

This function returns the font actually set.

SEE

vqt_name, vq_gdos

vst_height, vst_point

Set the text height

Class: VDI *Category: Text Attributes*

SYNOPSIS

```
#include <vdi.h>

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

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

DESCRIPTION

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

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

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

RETURNS

The function vst_point returns the height actually set in points.

SEE

vq_extnd

vst_load_fonts

Load GDOS fonts

Class: VDI *Category: Workstation Control*

SYNOPSIS

```
#include <vdi.h>

add=vst_load_fonts(handle,select);

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

DESCRIPTION

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

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

RETURNS

This function returns the number of additional fonts loaded.

SEE

v_opnvwk, v_opnwk, vq_gdos, vst_unload_fonts, vst_font, vqt_name

EXAMPLE

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

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

vst_rotation

Select rotated text

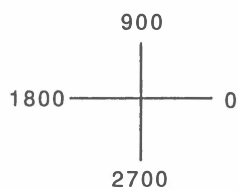
Class: VDI *Category: Text Attributes*

SYNOPSIS

```
#include <vdi.h>

set_angle=vst_rotation(handle,angle);

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

DESCRIPTION

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



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

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

RETURNS

This function returns the rotation angle actually set.

SEE

vq_extnd

vst_unload_fonts — Un-load GDOS fonts

Class: VDI *Category: Workstation Control*

SYNOPSIS

```
#include <vdi.h>

vst_unload_fonts(handle,select);

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

DESCRIPTION

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

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

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

SEE

v_opnvwk, v_opnwk, vq_gdos, vst_load_fonts

EXAMPLE

```
#include <vdi.h>

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

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

vswr_mode

Set graphics drawing mode

Class: VDI *Category: Graphics Attributes*

SYNOPSIS

```
#include <vdi.h>

new_mode=vswr_mode(handle,mode);

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

DESCRIPTION

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

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

RETURNS

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

SEE

v_opnvwk, v_opnwk

4 GEMDOS Library

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

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

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

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

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

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

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

Cauxin $03	Read a character from GEMDOS handle 2

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

SYNOPSIS

```
#include <osbind.h>

x=Cauxin();

short x;  character obtained from standard aux
```

DESCRIPTION

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

SEE

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

CAVEATS

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

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

Cauxis $12 Check status of GEMDOS handle 2

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

SYNOPSIS

```
#include <osbind.h>

x=Cauxis();

short x;      status of standard auxiliary input
```

DESCRIPTION

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

RETURNS

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

SEE

Cauxin, Bconin, Bconstat

CAVEATS

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

Cauxos $13 Check output status of GEMDOS handle 2

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

SYNOPSIS

```
#include <osbind.h>

x=Cauxos();

short x;      status of standard auxiliary output
```

DESCRIPTION

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

RETURNS

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

SEE

Cauxout, Bconout, Bcostat

CAVEATS

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

Cauxout $04 Write a character to GEMDOS handle 2

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

SYNOPSIS

```
#include <osbind.h>

Cauxout(x);

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

DESCRIPTION

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

SEE

Cconout, Cconin, Cconrs, Crawio, Cconis, Bconout

CAVEATS

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

Cconin $01 — Read a character from GEMDOS handle 0

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

SYNOPSIS

```
#include <osbind.h>

x=Cconin();

long x;    character obtained from standard in
```

DESCRIPTION

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

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

The non-ASCII keys (e.g. the function and cursor keys) return 0 for the ASCII value, so that the scan code is used to decipher them. The shift key status gives the state of the keyboard modifiers (Shift, Ctrl, Alt etc.) and are as described under the BIOS function Kbshift. Note that the shift key status is only returned if bit 3 in the system variable conterm (the character at 0x484) is set. This defaults to off.

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

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

RETURNS

As noted above.

SEE

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

CAVEATS

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

Cconis $0B Check status of standard input

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

SYNOPSIS

```
#include <osbind.h>

x=Cconis();

short x;       status of standard input
```

DESCRIPTION

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

RETURNS

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

SEE

Cconin, Bconin, Bconstat

Cconos $10 — Check status of standard output

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

SYNOPSIS

```
#include <osbind.h>

x=Cconos();

short x;        status of standard output
```

DESCRIPTION

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

RETURNS

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

SEE

Cconout, Bconout, Bcostat

Cconout $02 Write a character to GEMDOS handle 1

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

SYNOPSIS

```
#include <osbind.h>

Cconout(x);

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

DESCRIPTION

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

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

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

SEE

Cconin, Crawio, Crawcin, Cconws, Bconout

CAVEATS

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

Cconrs $0A — Read a string from standard input

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

SYNOPSIS

```
#include <osbind.h>

Cconrs(buf);

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

DESCRIPTION

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

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

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

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

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

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

RETURNS

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

SEE

Cconout, Cconis, Cconos, Bconin, Bconout

CAVEATS

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

Cconws $09 — Write string to standard output

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

SYNOPSIS

```
#include <osbind.h>

Cconws(x);

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

DESCRIPTION

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

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

SEE

Crawio, Cconout, Bconout

CAVEATS

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

Cnecin $08	Cooked input from standard in

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

SYNOPSIS

```
#include <osbind.h>

x=Cnecin();

long x      character obtained from standard in
```

DESCRIPTION

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

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

The non-ASCII keys (e.g. the function and cursor keys) return 0 for the ASCII value, so that the scan code is used to decipher them. The shift key status gives the state of the keyboard modifiers (Shift, Ctrl, Alt etc.) and are as described under the BIOS function Kbshift. Note that the shift key status is only returned if bit 3 in the system variable conterm is set. This defaults to off.

SEE

Crawio, Cconin, Crawcin, Cconrs, Cconis, Bconin

CAVEATS

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

Cprnos $// Check status of standard printer output

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

SYNOPSIS

```
#include <osbind.h>

x=Cprnos();

short x;      status of standard printer output
```

DESCRIPTION

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

RETURNS

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

SEE

Cconout, Bconout, Bcostat

Cprnout $05 Write a character to GEMDOS handle 3

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

SYNOPSIS

```
#include  <osbind.h>

status  =  Cprnout(x);

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

DESCRIPTION

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

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

RETURNS

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

SEE

Cconout, Bconout

Crawcin $07   Raw input from standard in

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

SYNOPSIS

```
#include <osbind.h>

x=Crawcin();

long x;      character obtained from standard in
```

DESCRIPTION

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

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

The non-ASCII keys (e.g. the function and cursor keys) return 0 for the ASCII value, so that the scan code is used to decipher them. The shift key status gives the state of the keyboard modifiers (Shift, Ctrl, Alt etc.) and are as described under the BIOS function Kbshift. Note that the shift key status is only returned if bit 3 in the system variable conterm is set. This defaults to off.

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

The SEE

Crawio, Cconin, Cnecin, Cconrs, Cconis, Bconin

CAVEATS

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

Crawio 806

Raw I/O to standard In/Out

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

SYNOPSIS

```
#include <osbind.h>

y=Crawio(x);

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

DESCRIPTION

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

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

The non-ASCII keys (e.g. the function and cursor keys) return 0 for the ASCII value, so that the scan code is used to decipher them. The shift key status gives the state of the keyboard modifiers (Shift, Ctrl, Alt etc.) and are as described under the BIOS function Kbshift. Note that the shift key status is only returned if bit 3 in the system variable conterm is set. This defaults to off.

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

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

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

SEE

Cconout, Cconin, Cconrs, Cconis, Bconout, Bconin

CAVEATS

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

$39 $3A

Dcreate, Ddelete

Create/Delete GEMDOS folder

Class: GEMDOS *Category: Directory Functions*

SYNOPSIS

```
#include <osbind.h>

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

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

DESCRIPTION

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

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

RETURNS

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

SEE

mkdir, rmdir

CAVEATS

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

Dfree $ 36   Get free disk space

Class: GEMDOS   *Category: Disk Functions*

SYNOPSIS

```
#include <osbind.h>

error = Dfree(info,drive);

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

DESCRIPTION

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

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

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

RETURNS

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

CAVEATS

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

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

Class: GEMDOS *Category: Disk Functions*

SYNOPSIS

```
#include <osbind.h>

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

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

DESCRIPTION

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

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

RETURNS

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

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

SEE

chgdsk, getdsk, Dgetpath, Dsetpath

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

Class: GEMDOS *Category: Directory Functions*

SYNOPSIS

```
#include <osbind.h>

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

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

DESCRIPTION

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

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

RETURNS

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

SEE

chdir, getcd, getcwd

CAVEATS

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

Fattrib $43 Get/Set file attributes

Class: GEMDOS *Category: File Manipulation*

SYNOPSIS

```
#include <osbind.h>

fa = Fattrib(fname,flag,attr);

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

DESCRIPTION

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

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

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

RETURNS

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

SEE

chgfa, getfa

CAVEATS

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

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

Fclose 83E Close GEMDOS file

Class: GEMDOS *Category: File Manipulation*

SYNOPSIS

```
#include <osbind.h>

error = Fclose(handle);

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

DESCRIPTION

This function closes the file associated with the specified handle.

RETURNS

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

SEE

Fcreate, Fopen, close

CAVEATS

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

Fcreate 83C Create or truncate a file

Class: GEMDOS *Category: File Manipulation*

SYNOPSIS

```
#include <osbind.h>

handle = Fcreate(name,attr);

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

DESCRIPTION

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

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

RETURNS

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

SEE

Fopen, Fclose, creat

CAVEATS

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

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

Fdatime 857 Get/Set file time stamp

Class: GEMDOS *Category: File Manipulation*

SYNOPSIS

```
#include <osbind.h>

error = Fdatime(timeptr,fh,flag);

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

DESCRIPTION

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

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

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

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

RETURNS

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

SEE

chgft, getft, ftunpk, ftpack

CAVEATS

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

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

Fdelete $41 Delete file

Class: GEMDOS *Category: File Manipulation*

SYNOPSIS

```
#include <osbind.h>

error = Fdelete(name);

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

DESCRIPTION

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

RETURNS

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

SEE

Ddelete, remove, unlink

CAVEATS

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

Fdup $45 — Duplicate standard file handle

Class: GEMDOS — *Category: File Manipulation*

SYNOPSIS

```
#include <osbind.h>

nh = Fdup(oh);

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

DESCRIPTION

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

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

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

RETURNS

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

SEE

Fforce, dup

CAVEATS

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

Fforce ẞ46 Redirect standard file handle

Class: GEMDOS *Category: File Manipulation*

SYNOPSIS

```
#include <osbind.h>

error = Fforce(stdh,nstdh);

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

DESCRIPTION

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

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

RETURNS

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

SEE

Fdup, dup2

EXAMPLE

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

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

$2F $1A

Fgetdta, Fsetdta

Get/Set data transfer address (DTA)

Class: GEMDOS *Category: File Manipulation*

SYNOPSIS

```
#include <osbind.h>

Fsetdta(dta);
dta = Fgetdta();

void *dta;          pointer to DTA
```

DESCRIPTION

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

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

SEE

Fsfirst, Fsnext, chgdta, getdta

Fopen $3D Open a GEMDOS file

Class: GEMDOS *Category: File Manipulation*

SYNOPSIS

```
#include <osbind.h>

handle = Fopen(name,mode);

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

DESCRIPTION

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

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

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

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

RETURNS

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

SEE

Fcreat, Fclose, open

Fread $3F

Read from an open file

Class: GEMDOS *Category: File Manipulation*

SYNOPSIS

```
#include <osbind.h>

len = Fread(handle,count,buf);

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

DESCRIPTION

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

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

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

RETURNS

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

SEE

Fcreat, Fclose, open

CAVEATS

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

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

Frename $56 Rename an existing file or folder

Class: GEMDOS *Category: File Manipulation*

SYNOPSIS

```
#include <osbind.h>

error = Frename(zero,old,new);

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

DESCRIPTION

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

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

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

RETURNS

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

SEE

rename

CAVEATS

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

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

Fseek $42

Seek to a new file position

Class: GEMDOS *Category: File Manipulation*

SYNOPSIS

```
#include <osbind.h>

apos = Fseek(rpos,handle,mode);

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

DESCRIPTION

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

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

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

RETURNS

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

SEE

_dseek, lseek

Fsfirst, Fsnext $4E, $4F

Find directory entry

Class: GEMDOS *Category: File Manipulation*

SYNOPSIS

```
#include <osbind.h>

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

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

DESCRIPTION

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

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

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

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

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

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

RETURNS

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

SEE

dfind, dnext, Fgetdta

EXAMPLE

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

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

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

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

Fwrite $40 — Write to an open file

Class: GEMDOS — *Category: File Manipulation*

SYNOPSIS

```
#include <osbind.h>

len = Fwrite(handle,count,buf);

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

DESCRIPTION

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

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

RETURNS

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

SEE

Fcreat, Fclose, Fread

CAVEATS

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

Malloc $48 Allocate a block of memory from the GEMDOS pool

Class: GEMDOS *Category: Memory Allocation*

SYNOPSIS

```
#include <osbind.h>

base = Malloc(amount);

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

DESCRIPTION

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

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

RETURNS

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

SEE

Mfree, Mshrink, malloc

CAVEATS

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

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

_mediach

Force media change on a logical device

Class: Lattice *Category: Device I/O*

SYNOPSIS

```
#include <osbind.h>

status=_mediach(dev);

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

DESCRIPTION

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

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

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

RETURNS

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

SEE

Getbpb, Mediach

Mfree 349 Release a block of memory to the GEMDOS pool

Class: GEMDOS *Category: Memory Allocation*

SYNOPSIS

```
#include <osbind.h>

error = Mfree(base);

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

DESCRIPTION

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

RETURNS

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

SEE

Malloc, Mshrink, free

Mshrink $4A

Shrink size of allocated block

Class: GEMDOS *Category: Memory Allocation*

SYNOPSIS

```
#include <osbind.h>

error = Mshrink(base,size);

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

DESCRIPTION

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

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

RETURNS

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

SEE

Malloc, Mfree, realloc

CAVEATS

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

Pexec $4B — Create/Execute process

Class: GEMDOS — *Category: Process Creation*

SYNOPSIS

```
#include  <osbind.h>

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

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

DESCRIPTION

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

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

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

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

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

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

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

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

RETURNS

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

SEE

Pterm0, Pterm, Ptermres, Mshrink

CAVEATS

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

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

Class: GEMDOS *Category: Process Creation*

SYNOPSIS

```
#include <osbind.h>

Pterm(ret);
Pterm0();

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

DESCRIPTION

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

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

RETURNS

The function does not (normally) return.

SEE

Pexec, Ptermres, Setexc, onbreak

Ptermres $31 — Terminate and stay resident (TSR)

Class: GEMDOS — *Category: Process Creation*

SYNOPSIS

```
#include <osbind.h>

Ptermres(keep,ret);

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

DESCRIPTION

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

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

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

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

RETURNS

The function does not (normally) return.

SEE

Pexec, Pterm, Setexc, onbreak

CAVEATS

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

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

Super $20 Get/Set/Inquire supervisor mode

Class: GEMDOS *Category: System Manipulation*

SYNOPSIS

```
#include <osbind.h>

oldssp = Super(stack);

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

DESCRIPTION

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

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

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

RETURNS

As noted above.

SEE

Supexec

CAVEATS

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

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

Sversion $30 Get GEMDOS version number

Class: GEMDOS *Category: System Manipulation*

SYNOPSIS

```
#include <osbind.h>

version = Sversion();

unsigned short version;   GEMDOS version number
```

DESCRIPTION

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

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

RETURNS

As noted above.

SEE

_tos, appl_init

EXAMPLE

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

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

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

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

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

Class: GEMDOS *Category: Date and Time*

SYNOPSIS

```
#include <osbind.h>

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

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

DESCRIPTION

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

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

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

RETURNS

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

SEE

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

CAVEATS

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

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

Class: GEMDOS *Category: Date and Time*

SYNOPSIS

```
#include <osbind.h>

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

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

DESCRIPTION

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

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

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

RETURNS

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

SEE

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

CAVEATS

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

5 BIOS Library

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

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

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

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

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

Bconin

Read a character from a device

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

SYNOPSIS

```
#include <osbind.h>

x=Bconin(dev);

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

DESCRIPTION

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

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

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

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

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

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

RETURNS

Returnerar när det finns en bokstav.

As noted above.

SEE

Bconstat, Cconin, Cauxin

CAVEATS

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

EXAMPLE

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

#include <osbind.h>

int oconterm;

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

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

int main(void)
{
  const char *unshift;

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

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

Bconout	Write a character to a device

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

SYNOPSIS

```
#include <osbind.h>

error=Bconout(dev,c);

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

DESCRIPTION

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

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

RETURNS

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

SEE

Bcostat, Cconout, Cauxout, Cprnout

Bconstat	Return device input status

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

SYNOPSIS

```
#include <osbind.h>

status=Bconstat(dev);

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

DESCRIPTION

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

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

RETURNS

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

SEE

Bconin, Cconis, Cauxis

Bcostat

Check character device output status

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

SYNOPSIS

```
#include <osbind.h>

status=Bcostat(dev);

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

DESCRIPTION

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

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

RETURNS

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

SEE

Bconout, Cconos, Cauxos, Cprnos

Drvmap

Return bitmap of mounted drives

Class: BIOS *Category: Device I/O*

SYNOPSIS

```
#include <osbind.h>

bmap=Drvmap();

unsigned long bmap;     bitmap of mounted drives
```

DESCRIPTION

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

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

RETURNS

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

SEE

Dsetdrv

EXAMPLE

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

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

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

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

Getbpb

Get BIOS parameter block for a device

Class: BIOS *Category: Device I/O*

SYNOPSIS

```
#include <osbind.h>

bpb=Getbpb(dev);

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

DESCRIPTION

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

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

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

RETURNS

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

SEE

_mediach

CAVEATS

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

Getmpb

Size machine memory

Class: BIOS *Category: Memory Allocation*

SYNOPSIS

```
#include <osbind.h>

Getmpb(mpb);

void *mpb;      pointer to prototype mpb
```

DESCRIPTION

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

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

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

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

SEE

Malloc

Kbshift

Find state of keyboard 'shift' keys

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

SYNOPSIS

```
#include <osbind.h>

state=Kbshift(mode);

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

DESCRIPTION

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

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

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

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

RETURNS

Kbshift returns the old state of the keyboard shift bits.

EXAMPLE

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

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

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

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

Mediach

Return media change status

Class: BIOS *Category: Device I/O*

SYNOPSIS

```
#include  <osbind.h>

status=Mediach(dev);

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

DESCRIPTION

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

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

RETURNS

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

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

SEE

Getbpb, _mediach

Rwabs, Lrwabs

Read/Write logical sectors on a device

Class: BIOS *Category: Device I/O*

SYNOPSIS

```
#include <osbind.h>

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

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

DESCRIPTION

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

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

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

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

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

RETURNS

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

SEE

Floprd, Flopwr

CAVEATS

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

Setexc

Set exception vector

Class: BIOS *Category: Vector Handling*

SYNOPSIS

```
#include <osbind.h>

old=Setexc(num,vec);

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

DESCRIPTION

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

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

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

RETURNS

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

Tickcal

Get system timer 'tick' interval

Class: BIOS *Category: Date and Time*

SYNOPSIS

```
#include <osbind.h>

tick=Tickcal();

long tick;    system tick interval
```

DESCRIPTION

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

RETURNS

As noted above.

6 XBIOS Library

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

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

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

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

Bioskeys

Reset keyboard translation tables

Class: XBIOS *Category: Keyboard Configuration*

SYNOPSIS

```
#include <osbind.h>

Bioskeys();
```

DESCRIPTION

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

SEE

Keytbl

Blitmode

Get/Set blitter configuration

Class: XBIOS *Category: Graphics Configuration*

SYNOPSIS

```
#include <osbind.h>

old=Blitmode(mode);

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

DESCRIPTION

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

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

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

RETURNS

As noted above.

EXAMPLE

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

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

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

Cursconf

Configure VT52 cursor

Class: XBIOS *Category: Graphics Configuration*

SYNOPSIS

```
#include <osbind.h>

old=Cursconf(function,rate);

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

DESCRIPTION

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

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

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

RETURNS

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

CAVEATS

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

Dosound

Initialise sound Dæmon

Class: XBIOS *Category: Sound Functions*

SYNOPSIS

```
#include <osbind.h>

Dosound(cmd);

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

DESCRIPTION

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

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

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

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

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

SEE

Giaccess

CAVEATS

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

Flopfmt

Format a track on a floppy disk

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

SYNOPSIS

```
#include <osbind.h>

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

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

DESCRIPTION

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

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

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

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

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

RETURNS

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

SEE

Floprd, Flopwr, Flopver, Floprate, Rwabs

CAVEATS

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

EXAMPLE

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

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

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

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

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

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

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

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

Floprate

Set floppy disk step rate

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

SYNOPSIS

```
#include <osbind.h>

old=Floprate(dev,rate);

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

DESCRIPTION

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

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

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

RETURNS

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

CAVEATS

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

Floprd

Read sectors from a floppy disk

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

SYNOPSIS

```
#include <osbind.h>

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

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

DESCRIPTION

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

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

RETURNS

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

SEE

Flopwr, Flopfmt, Flopver, Floprate, Rwabs

Flopver

Verify sectors on a floppy disk

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

SYNOPSIS

```
#include <osbind.h>

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

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

DESCRIPTION

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

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

RETURNS

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

SEE

Flopwr, Flopfmt, Floprd, Floprate, Rwabs

Flopwr

Write sectors to a floppy disk

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

SYNOPSIS

```
#include <osbind.h>

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

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

DESCRIPTION

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

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

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

RETURNS

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

SEE

Floprd, Flopfmt, Flopver, Floprate, Rwabs

Getrez

Find current screen mode

Class: XBIOS *Category: Graphics Configuration*

SYNOPSIS

```
#include <osbind.h>

res=Getrez();

short res;        current screen mode
```

DESCRIPTION

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

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

RETURNS

As noted above.

SEE

v_opnwk, Setscreen

CAVEATS

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

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

Gettime, Settime

Get/Set IKBD time

Class: XBIOS *Category: Date and Time*

SYNOPSIS

```
#include <osbind.h>

time=Gettime();
Settime(time);

long time;             IKBD time value
```

DESCRIPTION

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

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

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

RETURNS

Gettime returns the packed IKBD time.

Giaccess	Read/Write sound chip registers

Class: XBIOS *Category: Sound Functions*

SYNOPSIS

```
#include <osbind.h>

val=Giaccess(data,reg);

short val;          value of register
short data;         data to write into register
short reg;          register to get/set
```

DESCRIPTION

The Giaccess function is used access the ST sound chip. The register to consider is passed in reg and the new data value to be loaded passed in data. If reg has bit 7 clear (i.e. ANDed with 0x7f) then the setting of the register is not changed and the current value returned. The legal values for reg are:

0 1	Channel A frequency
2 3	Channel B frequency
4 5	Channel C frequency
6	Noise period
7	Enable flags
10	Channel A amplitude
11	Channel B amplitude
12	Channel C amplitude
13 14	Envelope period
15	Envelope shape

RETURNS

The function returns the new value of the register in val.

Ikbdws

Write string to keyboard processor

Class: XBIOS *Category: IKBD I/O*

SYNOPSIS

```
#include <osbind.h>

Ikbdws(count,buf);

short count;          number of bytes to write-1
const char *buf;      pointer to characters to write
```

DESCRIPTION

The Ikbdws function is used to write a string to the IKBD. count-1 characters are written from a buffer at buf.

SEE

Iorec, Initmous

Initmous

Set mouse mode and packet handler

Class: XBIOS *Category: IKBD I/O*

SYNOPSIS

```
#include <osbind.h>

Initmous(mode,param,hand);

short mode;              new mouse mode
void *param;             mouse mode parameter block
void (*hand)(void);      mouse packet handler
```

DESCRIPTION

Initmous is used to change the way the mouse movements are interpreted by the system. The mouse is capable of operating in several modes, the value of mode sets which one is to be used:

Value	Meaning
0	Disable mouse.
1	Enable relative mouse mode, i.e. report the position changes to the packet handler.
2	Enable absolute mouse mode, i.e. always report an absolute mouse position to the packet handler.
4	Enable mouse keycode mode, i.e. never send motion packets, but pretend that a cursor key was pressed.

If the mouse is being placed into relative or keycode mode, param should point to a structure of the form:

```
struct param
{
  char topmode;
  char buttons;
  char xparam;
  char yparam;
};
```

The topmode element can have two values; 0 indicates that Y=0 occurs at the bottom of the screen; 1 indicates that Y=0 occurs at the top of the screen.

buttons allows the button reporting state to be changed. If bit 2 is set then the mouse buttons act like normal keys, otherwise they are reported as packets to the handler. Bits 0 and 1 (when set) cause the absolute mouse position to be reported on pressing and/or on releasing a mouse button respectively.

xparam and yparam change the way the X and Y position information is reported. They have different meanings for each of the three mouse modes:

Mode	Meaning
Relative	Mouse threshold, the number of mouse 'clicks' between relative position reports.
Absolute	Mouse scaling factor, the number of 'clicks' to give a single step in the absolute position.
Keycode	Mouse delta factor, the number of 'clicks' before reporting a left/right/up/down cursor motion.

In mouse absolute mode the param structure is extended so that it has the form:

```
struct param
{
  char  topmode;
  char  buttons;
  char  xparam;
  char  yparam;
  short  xmax;
  short  ymax;
  short  xinitial;
  short  yinitial;
};
```

xmax and ymax specify the maximum X and Y positions that the mouse may be allowed to move to, whilst xinitial and yinital give the position at which the mouse should be placed.

hand points to a mouse packet handler which will be called when mouse packets become available. Note that in keycode mode you need not supply a handler.

SEE

Ikbdws, Kbdvbase

CAVEATS

If you are using the AES or VDI then changing the mode of the mouse from the relative mode required for their operation will stop them from functioning correctly.

Iorec

Find serial device I/O structure

Class: XBIOS *Category: MFP Configuration*

SYNOPSIS

```
#include  <osbind.h>

base=Iorec(dev);

void  *base;             base of I/O record
short  dev;              serial device
```

DESCRIPTION

Iorec is used to obtain the base of the system data structure for one of the serial devices. The parameter dev gives the device:

Value	Device
0	RS-232
1	Keyboard
2	MIDI

The structure returned has the form:

```
struct  iorec
{
  char  *ibuf;          pointer to buffer
  short  ibufsiz;       size of buffer
  short  ibufhd;        head index
  short  ibuftl;        tail index
  short  ibuflow;       low-water mark
  short  ibufhi;        high-water mark
};
```

If the structure requested was the for the RS-232 port then a second structure follows the first giving the RS-232 output buffer structure.

SEE

Midiws, Bconout, Bcostat, Bconin, Bconstat, Rsconf

Jenabint, Jdisint

Enable/Disable 68901 interrupt

Class: XBIOS *Category: MFP Configuration*

SYNOPSIS

```
#include <osbind.h>

Jdisint(intno);  disable MFP interrupt
Jenabint(intno); enable MFP interrupt

short intno;         interrupt to manipulate
```

DESCRIPTION

The Jenabint and Jdisint functions enable and disable respectively interrupt intno on the 68901. This function is most often with Mfpint to enable or disable interrupts after changing the handler. The values for intno are as described under Mfpint.

SEE

Mfpint

Kbdvbase — Obtain system IKBD/MIDI dispatch handler

Class: XBIOS *Category: IKBD/MIDI I/O*

SYNOPSIS

```
#include <osbind.h>

base=Kbdvbase();

void (*volatile *base)(void);  pointer to structure
```

DESCRIPTION

The Kbdvbase function obtains a pointer to the system structure used for dispatching MFP ACIA interrupts, so that you may patch into these if you wish. The Kbdvbase structure has the form:

```
struct kbdvecs
{
  void  (*midivec)(void);        MIDI-input
  void  (*vkbderr)(void);        keyboard error
  void  (*vmiderr)(void);        MIDI error
  void  (*statvec)(void);        IKBD status packet
  void  (*mousevec)(void);       mouse packet
  void  (*clockvec)(void);       clock packet
  void  (*joyvec)(void);         joystick packet
  void  (*midisys)(void);        system MIDI vector
  void  (*ikbdsys)(void);        system IKBD vector
  char  ikbdstate;               IKBD packet state
};
```

These vectors are used by the system for the following purposes:

midivec	MIDI input, by default a character is available in D0, which is then buffered into an Iorec structure.
vkbderr vmiderr	Keyboard and MIDI overrun handler.
statvec mousevec clockvec joyvec	IKBD status, mouse, clock and joystick packet handlers. These routines are passed a pointer to the received packet in A0.
midisys ikbdsys	Low-level MIDI and IKBD packet handlers. These routines are called initially and parse the status of the MFP before calling the appropriate sub-function.

If you replace any of the handlers you should either call the old handler or return via an RTS instruction.

RETURNS

As noted above.

SEE

Mfpint

Kbrate — Get/Set the keyboard repeat rate and delay

Class: XBIOS *Category: Keyboard Configuration*

SYNOPSIS

```
#include <osbind.h>

old=Kbrate(delay,rate);

short old;        packed old delay and repeat rate
short delay;      initial delay before repeat starts
short rate;       new repeat rate
```

DESCRIPTION

Kbrate is used to change the keyboard repeat rate and the initial delay before repeating starts. delay gives the time (in 50Hz system ticks) before the key starts repeating, whilst rate gives the rate at which the key is to repeat. If a parameter is -1 then the current value is not changed.

RETURNS

A packed word is returned giving the old key repeat and delay rates. The initial delay is in the high byte of old, whilst the repeat rate is in the low byte.

Keytbl

Change keyboard translation tables

Class: XBIOS *Category: Keyboard Configuration*

SYNOPSIS

```
#include <osbind.h>

ktab=Keytbl(normal,shift,caps);

char **ktab;            keyboard translation vector
const char *normal;     un-shifted translation table
const char *shift;      shifted translation table
const char *caps;       CAPS-lock translation table
```

DESCRIPTION

Keytbl is used to change the mapping from keyboard scan codes to key-presses. Note that *all* keyboards return identical scan-codes for keys in the same place, but it is these translation tables, which give the ASCII value for the legend marked on a key, that are used to internationalise a keyboard.

The normal, shift and caps pointers should point a arrays of 128 characters which map scan-codes into ASCII codes when the appropriate key is depressed. If a scan-code does not have an ASCII representation the value returned is 0.

If you do not wish to change one of the translation tables the value (char *)-1 should be passed.

RETURNS

Keytbl returns in ktab a pointer to the structure in which all three tables are held:

```
struct keytab
{
  char *unshift;     /* normal table */
  char *shift;       /* shifted table */
  char *capslock;    /* CAPS-lock table */
};
```

SEE

Bioskeys

Logbase

Find base of current drawing area

Class: XBIOS *Category: Graphics Configuration*

SYNOPSIS

```
#include <osbind.h>

base=Logbase();

void *base;       base of logical screen
```

DESCRIPTION

Logbase returns a pointer to the base of the logical screen (i.e. the one onto which any drawing by the GEM VDI is done).

Do not confuse the physical and logical screens. The physical screen is that displayed, whilst the logical screen is the one onto which drawing occurs. Normally these will be the same but this is not required.

RETURNS

The function returns the base of the logical screen.

SEE

Physbase, Setscreen

Mfpint

Set MFP interrupt handler

Class: XBIOS *Category: MFP Configuration*

SYNOPSIS

```
#include <osbind.h>

Mfpint(num,hand);

short num;                interrupt number to change
void (*hand)(void)        new interrupt handler
```

DESCRIPTION

Mfpint is used to change one of the multi-function peripheral adaptor (MFP) vectors. The vector to change is given by num, which has values:

Vector	Function
0	Parallel port
1	RS-232 Data Carrier Detect
2	RS-232 Clear-To-Send
3	BitBlt complete
4	RS-232 baud rate generator (Timer D)
5	200Hz System clock (Timer C)
6	Keyboard/MIDI
7	Floppy and Hard disk
8	Horizontal Blank (Timer B)
9	RS-232 transmit error
10	RS-232 transmit buffer empty
11	RS-232 receive error
12	RS-232 receive buffer full
13	DMA sound (Timer A)
14	RS-232 ring indicator
15	Mono monitor detect/DMA sound complete

The new interrupt handler is passed in hand. Note that installing a handler does not enable an interrupt this must be done separately via Jenabint.

SEE

Setexc, Jenabint, Jdisint

CAVEATS

The old MFP interrupt handler is discarded and so cannot subsequently be restored.

Note that the DMA sound option is only implemented on the Atari STE.

Midiws

Write string to MIDI port

Class: XBIOS *Category: MIDI I/O*

SYNOPSIS

```
#include  <osbind.h>

Midiws(count,buf);

short  count;           number  of  bytes  to  write-1
const  char  *buf;      pointer  to  characters  to  write
```

DESCRIPTION

The Midiws function is used to write a string to the MIDI port. count-1 characters are written from a buffer at buf.

SEE

Iorec

Ongibit, Offgibit

Atomically set/reset port A bit

Class: XBIOS *Category: Miscellaneous Functions*

SYNOPSIS

```
#include  <osbind.h>

Ongibit(onmask);
Offgibit(offmask);

short  onmask;        mask of bits to set
short  offmask;       mask of bits to clear
```

DESCRIPTION

Ongibit and Offgibit are used to atomically set and reset bits on the sound chip port A. This atomic access is *essential* as the BIOS often modifies these bits under interrupt control. For Ongibit, onmask contains a 1 in every bit position which is to be set and a 0 in every position which is to be unchanged. By comparison the Offgibit offmask contains a 1 in every bit position which is to be unchanged and a 0 in every position which is to be reset.

The bits in these masks are used for the following purposes:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Unused	General Purpose Output	Centronics Strobe	RS-232 DTR	RS-232 RTS	Floppy 1 Select	Floppy 0 Select	Floppy Side Select

SEE

Rsconf, Floprd, Flopwr

Physbase

Find base of current screen display

Class: XBIOS *Category: Graphics Configuration*

SYNOPSIS

```
#include <osbind.h>

base=Physbase();

void *base;       base of physical screen
```

DESCRIPTION

Physbase returns a pointer to the base of the physical screen (i.e. the one actually displayed).

Do not confuse the physical and logical screens. The physical screen is that displayed, whilst the logical screen is the one onto which drawing occurs. Normally these will be the same but this is not required.

RETURNS

The function returns the base of the physical screen.

SEE

Logbase, Setscreen

Protobt

Build prototype boot sector

Class: XBIOS *Category: Miscellaneous Functions*

SYNOPSIS

```
#include <osbind.h>

Protobt(buf,serial,type,exec);

void *buf;        512 byte prototype buffer
long serial;      serial number
short type;       disk type
short exec;       executable status of boot sector
```

DESCRIPTION

The Protobt function is used to build a boot sector for freshly formatted floppies. buf should point to a 512 byte buffer into which the sector will be built. This should contain any boot sector code you require.

serial gives the serial number to use for the disk. Note that the BIOS uses the serial number to distinguish floppies so if you give disks identical serial numbers they may become damaged. If serial has the value -1 then the current serial number in the boot sector is unchanged, otherwise if it has a value ≥0x01000000 then a random serial number is computed and used.

type specifies the disk type to construct it may have the values:

0	40 tracks, single sided (180K)
1	40 tracks, double sided (360K)
2	80 tracks, single sided (360K)
3	80 tracks, double sided (720K)
-1	Do not change type information

exec specifies whether the resulting sector is to executable. If exec is 0 the sector is made non-executable, 1 it is made executable and -1 the executable/non-executable status is preserved.

SEE

Flopfmt

Prtblk

Print bitmap

Class: XBIOS *Category: Printer Functions*

SYNOPSIS

```
#include <osbind.h>

status=Prtblk(blk);

short status                error status
void *blk;                  pointer to prtarg structure
```

DESCRIPTION

Prtblk is the general ST bitmap print utility. blk should point to a structure of the form:

```
struct prtarg
{
  char *blkptr;                      block pointer
  unsigned short offset;             bit offset
  unsigned short width;              width
  unsigned short height;             height
  unsigned short left;               left leader
  unsigned short right;              right trailer
  unsigned short srcres;             source resolution
  unsigned short dstres;             destination resolution
  unsigned short *colpal;            colour palette
  unsigned short type;               printer type
  unsigned short port;               printer port
  char *masks;                       halftone masks
};
```

The blkptr member points to the base of a bitmap to print, or to a string in text mode. offset gives the offset of the first bit to printed from the base of blkptr. height gives the height of the bitmap in pixels, or is 0 to indicate that this is a text mode usage. width gives the bitmap pixel width or a count of the number of characters to print in text mode. left and right specify the number of pixels to be skipped at the left and right hand edges when moving between lines.

type may have 1 of 4 values indicating the type of printer. The current values are:

0	Monochrome Atari printer
1	Colour Atari printer
2	Monochrome Daisy-wheel
3	Monochrome Epson Compatible

srcres gives the source resolution using the same values as Getrez. dstres gives the printer resolution and is 0 for draft mode and 1 for final mode. colpal points to a list of the colour palette settings. port gives the port to use, 0 for parallel, 1 for serial. masks points to a set of half-tone masks to use when mapping colours onto printer colours, or NULL to use the default masks.

Note that the system global _prt_cnt should be set to 1 prior to calling this function to ensure that the user cannot hit Alt-Help.

RETURNS

Prtblk returns zero if the printing was completed successfully or a negative error code.

```
/* Emulate the Scrdmp() command */
#include <osbind.h>
#include <stdlib.h>
#include <linea.h>

enum {MONO_ATARI, COLOUR_ATARI, DAISY, EPSON};

int lock(void)
{
  *(short *)0x4ee=1;  /* lock out Alt-Help */
}

int main(void)
{
  static struct
  {
    char *blkptr;
    unsigned short offset,width,height,left,right;
    unsigned short srcres,dstres,*colpal,type,port;
    char *masks;
  } prt;
  short palette[16],conf;
  register int i;

  conf=Setprt(-1);
  prt.blkptr=Physbase();   /* dump physical screen */
  if (conf&1)
    abort();               /* can't do daisywheels */
  else if (conf&4)
    prt.type=EPSON;
  else if (conf&2)
    prt.type=COLOUR_ATARI;
  else
    prt.type=MONO_ATARI;
  for (i=16; i--; )
    palette[i]=Setcolor(i,-1);
  prt.colpal=palette;      /* get palette */
  prt.port=(conf&16)>>4;   /* port */
  prt.srcres=Getrez();     /* screen resolution */
  prt.dstres=(conf&8)>>3;  /* printer resolution */
  linea0();                /* init Line-A for _MAX */
  prt.width=V_X_MAX;       /* find screen width */
  prt.height=V_Y_MAX;      /* and height */
  Supexec(lock);           /* enable Alt-Help */
  return Prtblk(&prt);     /* and dump */
}
```

Puntaes

Discard AES

Class: XBIOS *Category: Miscellaneous Functions*

SYNOPSIS

```
#include <osbind.h>

Puntaes();
```

DESCRIPTION

Puntaes is used to throw away the AES and any memory it occupies. Note that this function will only work for RAM-loaded TOS.

Random

Obtain random number

Class: XBIOS *Category: Miscellaneous Functions*

SYNOPSIS

```
#include <osbind.h>

rand=Random();

long rand;              system random value
```

DESCRIPTION

Rand is the system random number generator and is normally used when obtaining serial numbers for freshly formatted floppies.

RETURNS

Random returns a 24 bit random number. Note that the algorithm used gives an *exact* 50% distribution for bit 0 and so this function should be used with care.

SEE

Protobt

Rsconf	Configure RS-232 communications port

Class: XBIOS *Category: MFP Configuration*

SYNOPSIS

```
#include <osbind.h>

save=Rsconf(speed,flow,ucr,rsr,tsr,scr);

unsigned long old;     old 68901 configuration
short speed;           new RS-232 speed request
short flow;            flow control mode
short ucr;             USART control register
short rsr;             receive status register
short tsr;             transmit status register
short scr;             synchronous character register
```

DESCRIPTION

Rsconf is used to configure the RS-232 communications interface. The speed parameter gives the requested speed:

Value	Baud Rate
0	19200
1	9600
2	4800
3	3600
4	2400
5	2000
6	1800
7	1200

Value	Baud Rate
8	600
9	300
10	200
11	150
12	134
13	110
14	75
15	50

flow allows the flow control method to be adjusted. The values are:

Value	Method
0	No flow control (default)
1	XON/XOFF (^S/^Q)
2	RTS/CTS
3	XON/XOFF and RTS/CTS

ucr sets the USART control register, the low byte only is used:

Bit 7	Bits 6-5	Bits 4-3	Bit 2	Bit 1	Bit 0
CLK/16	00-8 bits per word 01-7 bits per word 10-6 bits per word 11-5 bits per word	00-No Start/Stop 01-1 Start,1 Stop 10-1 Start,$1\frac{1}{2}$Stop 11-1 Start, 2 Stop	Parity	Use odd parity	Unused

rsr sets the receiver status register, the low byte only is used:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Buffer full	Overrun error	Parity error	Frame error	Break detect	Match busy	Sync strip	Receiver enable

tsr sets the transmit status register, the low byte only is used:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Buffer empty	Underrun error	Parity error	Frame error	Break detect	Match busy	Sync strip	Receiver enable

scr sets the synchronous character register, the low byte only is used and gives the character that will be searched for when an underrun error occurs in synchronous mode.

If any of the parameters has the value -1 then it is ignored and the current setting is unchanged.

RETURNS

Rsconf returns the old 68901 settings in a long word with the old ucr, rsr, tsr and scr packed from high to low in that order.

SEE

Bconout, Bcostat, Bconin, Bconstat, Ioctl

Scrdmp

Copy screen to printer

Class: XBIOS *Category: Printer Functions*

SYNOPSIS

```
#include <osbind.h>

Scrdmp();
```

DESCRIPTION

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

SEE

Prtblk, v_hardcopy

Setcolor

Set display palette

Class: XBIOS *Category: Graphics Configuration*

SYNOPSIS

```
#include <osbind.h>

old=Setcolor(num,new);

short old;   old BCD colour value
short num;   logical colour number to modify
short new;   new BCD colour value
```

DESCRIPTION

Setcolor is used to change the mapping from logical to physical colours. Colour values are stored in a BCD manner with the least-significant bit replacing the most-significant bit. A physical colour is packed in the following manner:

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

R0 represents the least-significant bit of the red component of the colour, R3 the most-significant. Similarly, G0-G3 give the green component and B0-B3 the blue component.

Note that the peculiar packing method is to ensure backward compatibility from the Atari STE to the Atari ST, hence bits R0, G0 and B0 are not used on the ST.

The logical colour to change is passed in num, and the new packed colour in new. If new has the value -1 then the colour is not changed.

RETURNS

Setcolor returns the old BCD value for the logical colour.

SEE

Setpalette

Setpalette

Set display palette

Class: XBIOS *Category: Graphics Configuration*

SYNOPSIS

```
#include <osbind.h>

Setpalette(palette);

short *palette;  pointer to screen palette
```

DESCRIPTION

Setpalette is used to reset the screen palette. For the current screen modes palette should point to an array of 16 words giving the BCD representations of the required screen colours.

Note that the palette assignment does not occur until the next vertical blank so a call to this routine should be followed by one to Vsync to ensure that the memory used by palette cannot be re-allocated before the new palette is installed.

SEE

Setcolor

CAVEATS

This function was spelt Setpallete (sic) in the original Atari bindings; both versions are included in the osbind.h file.

Setprt — Set/Get printer configuration

Class: XBIOS *Category: Printer Functions*

SYNOPSIS

```
#include <osbind.h>

old=Setprt(new);

short old;        old configuration word
short new;        new configuration word
```

DESCRIPTION

Setprt is used to get or set the printer configuration. The configuration is changed to the value of new, currently 6 bits are defined in this:

Bit Number	Meaning when clear	Meaning when set
0	Dot matrix	Daisy wheel
1	Monochrome	Colour
2	Atari mode	"Epson" compatible
3	Preview mode	Final mode
4	Parallel port	RS-232 port
5	Continuous	Single sheet

Other bits should be preserved for future compatibility. In order to read the current status the value -1 may be used for new, in which case the configuration is not changed and the current configuration returned.

RETURNS

Setprt returns the old printer configuration word.

SEE

Scrdmp, Prtblk

CAVEATS

Beware of some older documentation which lists bit 1 as being set for mono and clear for colour, the bit should be *clear* for mono and *set* for colour.

Setscreen	Set screen parameters

Class: XBIOS *Category: Graphics Configuration*

SYNOPSIS

```
#include <osbind.h>

Setscreen(log,phys,mode);

void *phys;      pointer to new physical screen base
void *log;       pointer to new logical screen base
short mode;      new screen mode request
```

DESCRIPTION

The Setscreen call is used to change the current screen mode, physical screen base and/or logical screen base. If any of the parameters is negative (e.g. -1) then that parameter is unchanged as a result of this call.

phys specifies a new physical screen base. This takes effect immediately (not at the next vertical blank as mentioned in older documentation), and as such you should be aware that screen 'flicker' may result. Note that on the Atari ST this *must* be on a 256 byte boundary. On the Atari STE this limitation has been relaxed and phys need only be word aligned.

log specifies a new logical screen base. It is onto this screen that all drawing is done. Note that it is recommended that after changing the logical screen base the (logical) screen be cleared to ensure that all pointers used internally by the VDI are correctly initialised.

mode specifies a new screen resolution. This parameter has the same values as those returned by Getrez. When mode is positive (i.e. the resolution is changed) the screen is automatically cleared and the internal state of the VT52 emulator reset.

SEE

Getrez

CAVEATS

This function does not inform the AES of a resolution change and so cannot be used once the AES has been initialised, unless you no longer require its services.

Ssbrk

Reserve system memory

Class: XBIOS *Category: Memory Allocation*

SYNOPSIS

```
#include <osbind.h>

base=Ssbrk(len);

void *base;        base of memory allocated
short len;         amount of memory required
```

DESCRIPTION

Ssbrk was provided on the very first STs to provide a way of reserving system memory before the OS was loaded from disk. It is no longer implemented or required.

Supexec

Execute function in supervisor mode

Class: XBIOS *Category: Miscellaneous Functions*

SYNOPSIS

```
#include <osbind.h>

val=Supexec(func);

long func();            function to call
```

DESCRIPTION

Supexec is used to call the named function in supervisor mode. The function should be careful if it wishes to call the BIOS or XBIOS since these are only re-entrant to three levels.

The value returned from the func is passed back as the return value from Supexec.

RETURNS

As noted above.

Vsync

Wait for vertical sync to occur

Class: XBIOS *Category: Miscellaneous Functions*

SYNOPSIS

```
#include <osbind.h>

Vsync();
```

DESCRIPTION

Vsync is used to wait for a vertical blank to occur. It is often used to prevent 'flicker' when drawing graphics or to ensure that vertical blank driven objects are complete before being re-used (e.g. Setpalette).

Xbtimer

Configure MFP timer

Class: XBIOS *Category: MFP Configuration*

SYNOPSIS

```
#include  <osbind.h>

Xbtimer(timer,ctrl,data,hand);

short  timer;             number of timer to change
short  ctrl;              value to place in control
                          register
short  data;              value for timer data register
void  (*hand)(void);      pointer to interrupt handler
```

DESCRIPTION

Xbtimer allows the 68901 timers to be setup. The timer to change is passed in timer and has a value 0-3 indicating timer A, B, C or D. control is placed in the control register of the timer. data is placed in the data register of the timer. The interrupt handler for the timer is pointed to hand. The allocation of the timers is:

Timer	Usage
A	DMA sound counter
B	HBlank counter
C	200Hz System timer
D	RS-232 baud rate generator

SEE

Mfpint

7 Line-A Library

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

The Line-A emulator provides the graphics primitives which are used by the VDI. The Line-A interface is in general inconsistent, difficult to use and completely non-portable. The name Line-A comes from the special 68000 instructions used to access the routines, which have the top nybble set to 'A'.

Before any of the Line-A routines may be used the linea0 function must be called to initialise the structures used by the bindings. All access to the Line-A routines is through a parameter block in which the input variables are placed, prior to executing the function, with a second data block made available for configuration and interrogation of the screen device layout.

The sixteen functions available in the Line-A are:

linea0	Initialise Line-A data structure
linea1	Plot single pixel
linea2	Get pixel value
linea3	Draw arbitrary line
linea4	Draw horizontal line
linea5	Render a filled rectangle
linea6	Render a line of a filled polygon
linea7	Perform a BITBLiT
linea8	Render bit-mapped character on screen
linea9	Show mouse cursor
lineaa	Hide mouse cursor
lineab	Transform mouse cursor
lineac	Remove user sprite
linead	Render user sprite
lineae	Copy raster form
lineaf	Flood fill area

linea0

Initialise Line-A data structure

Class: Line-A *Category: Initialisation*

SYNOPSIS

```
#include <linea.h>

data=linea0();

struct la_data *data;     pointer to Line-A structure

extern LINEA_INFO la_info;
```

DESCRIPTION

linea0 is used to initialise the structure used when interrogating the Line-A data structures and calling the Line-A routines. It fills in the external structure la_info, containing the items:

```
typedef struct linea_info
{
  long li_d0;                    linea data structure
  struct la_data *li_a0;         linea data structure
  struct la_font **li_a1;        system font vector
  long (*li_a2)();               linea function vector
} LINEA_INFO;
```

li_d0 and li_a0 both point to the middle of the Line-A structures. Positive offsets from them are input parameters to Line-A commands, whilst negative offsets give the configuration and status information. The positive offset structure is:

```
typedef struct la_data
{
  short ld_vplanes;       number of bit planes
  short ld_vwrap;         number of bytes/video line
  short *ld_contrl;       pointer to CONTRL array
  short *ld_intin;        pointer to INTIN array
  short *ld_ptsin;        pointer to PTSIN array
  short *ld_intout;       pointer to INTOUT array
  short *ld_ptsout;       pointer to PTSOUT array
  short ld_colbit[4];     colour bit-plane[i] value
  short ld_lstlin;        draw last pixel flag
  short ld_lnmask;        line-style mask
  short ld_wmode;         writing mode
  short ld_x1;            X1 coordinate
  short ld_y1;            Y1 coordinate
  short ld_x2;            X2 coordinate
  short ld_y2;            Y2 coordinate
  short *ld_patptr;       fill pattern pointer
  short ld_patmsk;        fill pattern mask
  short ld_mfill;         multi-plane fill flag
  short ld_clip;          clipping flag
  short ld_xmincl;        minimum X clipping value
  short ld_ymincl;        minimum Y clipping value
  short ld_xmaxcl;        maximum X clipping value
```

```
    short  ld_ymaxcl;       maximum Y clipping value
    short  ld_xdda;         accumulator for textblt dda
    short  ld_ddainc;       fixed point scale factor
    short  ld_scaldir;      scale direction flag
    short  ld_mono;         current font is monospaced
    short  ld_srcx;         X coord of character in font
    short  ld_srcy;         Y coord of character in font
    short  ld_dstx;         X coord of character on screen
    short  ld_dsty;         Y coord of character on screen
    short  ld_delx;         width of character
    short  ld_dely;         height of character
    void  *ld_fbase;        pointer to start of font form
    short  ld_fwidth;       width of font form
    short  ld_style;        textblt special effects flags
    short  ld_litemsk;      lightening mask
    short  ld_skewmsk;      skewing mask
    short  ld_weight;       thickening factor
    short  ld_roff;         skew offset above baseline
    short  ld_loff;         skew offset below baseline
    short  ld_scale;        scaling flag
    short  ld_chup;         character rotation angle
    short  ld_textfg;       text foreground colour
    void  *ld_scrtchp;      word-aligned effects buffer
    short  ld_scrpt2;       offset to scaling buffer
    short  ld_textbg;       text background colour
    short  ld_copytran;     copy raster form type flag
    int  (*ld_seedabort)(void);
                            seedfill abort detect
  } LA_DATA;
```

The negative offset structure is:

```
  typedef struct la_ext
  {
    long  ld_resvd1;
    struct  la_font  *ld_cur_font;
                                pointer to current font
                                header
    short  ld_resvd2[23];
    short  ld_m_pos_hx;         mouse x hot spot
    short  ld_m_pos_hy;         mouse y hot spot
    short  ld_m_planes;         writing mode for mouse
    short  ld_m_cdb_bg;         mouse background colour
    short  ld_m_cdb_fg;         mouse foreground colour
    short  ld_mask_form[32];    mouse mask and form
    short  ld_inq_tab[45];      vq_extnd information
    short  ld_dev_tab[45];      v_opnwk information
    short  ld_gcurx;            current mouse x position
    short  ld_gcury;            current mouse x position
    short  ld_m_hid_ct;         mouse hide count
    short  ld_mouse_bt;         mouse button status
    short  ld_req_col[3][16];   internal vq_color lookup
    short  ld_siz_tab[15];      current text, line and
                                marker sizes
    short  ld_resvd3;
    short  ld_resvd4;
    short  *ld_cur_work;        current vwork attributes
    struct  la_font  *ld_def_font;
                                default font header
    struct  la_font  *ld_font_ring[4];
                                vdi font ring
    short  ld_font_count;       number of fonts in font
                                ring
```

```
        short  ld_resvd5[45];
        unsigned  char  ld_cur_ms_stat;
                                   mouse  status
        char  ld_resvd6;
        short  ld_v_hid_cnt;       cursor  hide  count
        short  ld_cur_x;           mouse  x  position
        short  ld_cur_y;           mouse  y  position
        char  ld_cur_flag;         mouse  draw  status
        char  ld_mouse_flag;       mouse  processing  enabled
        long  ld_resvd7;
        short  ld_v_sav_xy[2];     saved  cursor  xy  position
        short  ld_save_len;        height  of  saved  form
        short  *ld_save_addr;      screen  address  of  saved
                                   form
        short  ld_save_stat;       save  status
        long  ld_save_area[4][16]; form  save  area
        void  (*ld_user_tim)();    user  timer  vector
        void  (*ld_next_tim)();    next  timer  vector
        void  (*ld_user_but)();    user  button  vector
        void  (*ld_user_cur)();    user  cursor  vector
        void  (*ld_user_mot)();    user  motion  vector
        short  ld_cel_ht;          cell  height
        short  ld_cel_mx;          max  x  cells
        short  ld_cel_my;          max  y  cells
        short  ld_cel_wr;          displacement  to  next
                                   vertical  cell
        short  ld_col_bg;          background  colour  index
        short  ld_col_fg;          foreground  colour  index
        void  *ld_cur_ad;          cursor  address
        short  ld_cur_off;         offset  to  first  cell
        short  ld_cur_xy[2];       cursor  xy  position
        char  ld_cur_cnt;          cursor  flash  period
        char  ld_cur_tim;          cursor  flash  countdown
        void  *ld_fnt_ad;          address  of  font  data
        short  ld_fnt_nd;          last  ade  in  font
        short  ld_fnt_st;          first  ade  in  font
        short  ld_fnt_wr;          font  form  width
        short  ld_x_max;           horizontal  pixel
                                   resolution
        void  *ld_off_ad;          pointer  to  font  offset
                                   table
        short  ld_status;          cursor  status
        short  ld_y_max;           vertical  pixel  resolution
        short  ld_bytes_lin;       width  of  destination  form
    }  LA_EXT;
```

Note that this structure may be accessed using ((LA_EXT *)la_info.li_a0-1)->ld..

The remaining structure members in linea_info are; li_a1 which points to a NULL terminated array of system fonts, *currently* three fonts are available. li_a2 points to an array of the 16 Line-A entry points so that you may remove the Line-A handler overhead and call them directly. If you do this be aware that some of the functions *must* be run in supervisor mode and that the registers they destroy is *completely* undefined.

To simplify access to these variables macros are provided to perform all the indirections. These macros are named on a variant of the structure names, so that to gain access to, for instance, ld_y_max you may simply use V_Y_MAX, or to access one of the positive structures, e.g. ld_patptr, simply PATPTR.

You should be aware that the CONTRL, INTIN, PTSIN, INTOUT and PTSOUT are inherited from the last user, hence if a process has terminated these arrays may point to non-allocated memory. If you need to use these arrays you should ensure that you have *either* allocated a VDI virtual workstation, *or* have placed pointers to your own private arrays in these elements.

The system fonts use the same format as GDOS fonts, a structure of the form:

```
typedef struct la_font
{
  short font_id;                    face identifier
  short font_size;                  font size in points
  char font_name[32];               face name
  short font_low_ade;               lowest ASCII value
  short font_hi_ade;                highest ASCII value
  short font_top_dst;               top line distance
  short font_ascent_dst;            ascent line distance
  short font_half_dst;              half line distance
  short font_descent_dst;           descent line distance
  short font_bottom_dist;           bottom line distance
  short font_fatest;                widest char in font
  short font_fat_cell;              widest char cell in font
  short font_left_off;              left offset
  short font_right_off;             right offset
  short font_thickening;            pixels to widen chars
  short font_underline;             underline pixel width
  short font_lightening;            lightening mask
  short font_skewing;               skewing mask
  short font_flags;                 flags
  short *font_horiz_off;            pointer to HOT
  short *font_char_off;             pointer to COT
  void *font_data;                  pointer to font form
  short font_width;                 font width
  short font_height;                font height
  struct la_font *font_next;        pointer to next font
} LA_FONT;
```

Most fields in the LA_FONT structure are self-explanatory with reference to v_gtext, the other fields are:

font_thickening	Number of pixels to increase each horizontal pixel run by to achieve a **bold** font.
font_underline	Number of pixels in the underline effect.
font_lightening	Mask used when removing pixels to create a 'disabled' character. This normally has the value 0x5555, indicating that alternate pixels should be dropped.
font_skewing	Mask used when creating *skewed* characters. This mask is considered rotated vertically, and then for each row that has the skew mask set the pixel row is shifted right by one pixel . The usual value is 0x5555, giving a skew of 26.6°.

font_flags	This consists of a bitmap giving flags for this font: Bit — Meaning (When set) 0 — Font is default system font 1 — Horizontal offset table present 2 — Font is in Motorola format 3 — Font is monospaced Note that *all* fonts which are in memory will be in Motorola format.
font_horiz_off	Pointer to horizontal offset table (HOT). This is an array of short integers with the most significant (signed) byte giving the left offset (i.e. added *prior* to printing) and the least significant (signed) byte giving the right offset (i.e. added *after* to printing). This can be useful for kerning or accented use. Note that the VDI output functions do not support horizontal offset tables correctly, and the Line-A routines are the only way to use them successfully.
font_char_off	Pointer to character offset table (COT). This is an array of shorts giving the 'X' co-ordinate of each character in the font within the form. Note that the first element is for the first character in the set and not 0, hence you must subtract font_low_ade before indexing into this array.

SEE

v_opnwk, v_opnvwk

linea1

Plot single pixel

Class: Line-A *Category: Pixel Manipulation*

SYNOPSIS

```
#include <linea.h>

linea1();

putpixel(x,y,colour);

INTIN[0]=colour;             colour of pixel to plot
PTSIN[0]=X;                  X co-ordinate of pixel
PTSIN[1]=Y;                  Y co-ordinate of pixel
```

DESCRIPTION

linea1 plots single pixels on screen. INTIN(0) holds the colour to give the pixel, PTSIN(0) and PTSIN(1) hold the required X and Y co-ordinates.

The putpixel macro is provided in linea.h to simplify the use of this function and takes parameters x, y and colour.

SEE

v_pmarker, v_pline, linea2, linea3, linea4

CAVEATS

This function pays no regard to any clipping rectangle installed in the Line-A input array.

linea2

Get pixel value

Class: Line-A *Category: Pixel Manipulation*

SYNOPSIS

```
#include <linea.h>

colour=linea2();

colour=getpixel(x,y);

short colour;                 colour of pixel
PTSIN[0]=X;                   X co-ordinate of pixel
PTSIN[1]=Y;                   Y co-ordinate of pixel
```

DESCRIPTION

linea2 obtains the colour value of a single pixels on screen. PTSIN(0) and PTSIN(1) hold the required X and Y co-ordinates, and the current value of the pixel is returned in colour.

The getpixel macro is provided in linea.h to simplify the use of this function and takes parameters (x, y) returning colour.

SEE

v_get_pixel, linea1

CAVEATS

This function pays no regard to any clipping rectangle installed in the Line-A input array.

linea3	Draw arbitrary line

Class: Line-A *Category: Line Drawing*

SYNOPSIS

```
#include <linea.h>

linea3();

X1=x1;                          starting X co-ordinate
Y1=y1;                          starting Y co-ordinate
X2=x2;                          ending X co-ordinate
Y2=y2;                          ending Y co-ordinate
COLBIT0=colour;                 value for bit plane 0
COLBIT1=colour>>1;              value for bit plane 1
COLBIT2=colour>>2;              value for bit plane 2
COLBIT3=colour>>3;              value for bit plane 3
LNMASK=style;                   line pattern mask.
WMODE=mode;                     writing mode.
LSTLIN=last;                    draw last pixel flag
```

DESCRIPTION

linea3 draws a line between points (X1,Y1) and (X2,Y2). The colour to use is split into bits and provided in the COLBIT elements. LNMASK gives the bit pattern to use when drawing the line, whilst the drawing mode is given by WMODE. The values for WMODE (which are the VDI MD_... modes -1) are:

0	Replace mode; the new data replaces the old.
1	Transparent mode only affects pixels where the pixel is already set.
2	Exclusive OR mode.
3	Reverse transparent mode only affects pixels where the source pixel is not set.

LSTLIN is used when drawing lines in XOR mode and normally is -1 indicating that the last point in the line is to be omitted, or if 0 the point is plotted.

SEE

linea1, linea4, v_pline, vswr_mode

CAVEATS

This function pays no regard to any clipping rectangle installed in the Line-A input array.

linea4

Draw horizontal line

Class: Line-A *Category: Line Drawing*

SYNOPSIS

```
#include <linea.h>

linea4();

X1=x1;                      starting X co-ordinate
X2=x2;                      ending X co-ordinate
Y1=y;                       Y co-ordinate
COLBIT0=colour;             value for bit plane 0
COLBIT1=colour>>1;          value for bit plane 1
COLBIT2=colour>>2;          value for bit plane 2
COLBIT3=colour>>3;          value for bit plane 3
WMODE=mode;                 writing mode
PATPTR=pattern;             pointer to fill pattern
PATMSK=index;               pattern count
MFILL=flag;                 multi plane fill flag
```

DESCRIPTION

linea4 draws a horizontal line between points (X1,Y1) and (X2,Y1). The colour to use is split into bits and provided in the COLBIT elements. PATPTR points to an array of PATMSK+1 line patterns. The pattern chosen for a particular line segment is then a function of Y1 and PATMASK. If MFILL is zero then the writing mode WMODE is used as described under linea3.

When MFILL is non-zero the value of WMODE is ignored and the planes are simply filled with the bits in COLBITs.

SEE

v_pline, linea1, linea3, linea5

CAVEATS

This function pays no regard to any clipping rectangle installed in the Line-A input array.

linea5 — Render a filled rectangle

Class: Line-A — *Category: Area Filling*

SYNOPSIS

```
#include  <linea.h>

linea5();

X1=x1;                      left X co-ordinate
Y1=y1;                      top Y co-ordinate
X2=x2;                      right X co-ordinate
Y2=y2;                      bottom Y co-ordinate
COLBIT0=colour;             value for bit plane 0
COLBIT1=colour>>1;          value for bit plane 1
COLBIT2=colour>>2;          value for bit plane 2
COLBIT3=colour>>3;          value for bit plane 3
WMODE=mode;                 writing mode
PATPTR=pattern;             pointer to fill pattern
PATMSK=index;               pattern count
MFILL=flag;                 multi plane fill flag
CLIP=state;                 clipping flag
XMINCL=x1clip;              left edge X clipping
YMINCL=y1clip;              top edge Y clipping
XMAXCL=x2clip;              right edge X clipping
YMAXCL=y2clip;              bottom edge Y clipping
```

DESCRIPTION

linea5 draws a filled rectangle with upper left corner (X1,Y1) and lower right corner (X2,Y1).

The COLBIT, PATPTR, PATMSK, MFILL and WMODE parameters are as described under linea4. Note that the PATPTR value is identical to that of linea4 which is used as the primitive for this function.

An optional clipping rectangle may be specified with this function; it has top left corner (XMINCL, YMINCL) and bottom right corner (XMAXCL, YMAXCL). To enable clipping CLIP should be set to 1, or disabled by setting CLIP to 0.

SEE

linea1, linea4, v_bar, v_recfl

linea6	Render a line of a filled polygon

Class: Line-A *Category: Line Drawing*

SYNOPSIS

```
#include <linea.h>

linea6();

PTSIN[]=...;                   array of vertices
CONTRL[1]=n;                   number of vertices
Y1=y1;                         line to draw
COLBIT0=colour;                value for bit plane 0
COLBIT1=colour>>1;             value for bit plane 1
COLBIT2=colour>>2;             value for bit plane 2
COLBIT3=colour>>3;             value for bit plane 3
WMODE=mode;                    writing mode
PATPTR=pattern;                pointer to fill pattern
PATMSK=index;                  pattern count
MFILL=flag;                    multi plane fill flag
CLIP=state;                    clipping flag
XMINCL=x1clip;                 left edge X clipping
YMINCL=y1clip;                 top edge Y clipping
XMAXCL=x2clip;                 right edge X clipping
YMAXCL=y2clip;                 bottom edge Y clipping
```

DESCRIPTION

linea6 draws one line of a filled polygon. The polygon is specified as an array of vertices in PTSIN, with the number of vertices in CONTRL(1). Note that the first vertex must be repeated as the last vertex, but this extra vertex is not included in the vertex count. The line drawn as a result of this function is Y1.

The COLBIT, PATPTR, PATMSK, MFILL and WMODE parameters are as described under linea4. Note that the PATPTR value is identical to that of linea4 which is used as the primitive for this function.

An optional clipping rectangle may be specified with this function; it has top left corner (XMINCL, YMINCL) and bottom right corner (XMAXCL, YMAXCL). To enable clipping CLIP should be set to 1, or disabled by setting CLIP to 0.

SEE

linea1, linea4, v_fillarea

CAVEATS

This function only performs the fill line selection correctly when the fill pattern height is an exact power of 2. Also you must ensure that the PTSIN array is large enough for your requirements, otherwise the system may crash mysteriously.

EXAMPLE

```
/*
 * draw a simple polygon filled with a single plane
 * pattern
 */

#include <linea.h>

int main(void)
{
  short pts[]={160,100,0,50,319,199,319,50,160,100};
  short contrl[2];
  short pattern[]=
  {
    0x0940,    /* 0000100101000000 */
    0x0940,    /* 0000100101000000 */
    0x0f40,    /* 0000111101000000 */
    0x0940,    /* 0000100101000000 */
    0x0940,    /* 0000100101000000 */
    0x0000,    /* 0000000000000000 */
    0x64dc,    /* 0110010011011100 */
    0x8a88,    /* 1000101010001000 */
    0xcac8,    /* 1100101011001000 */
    0x2a88,    /* 0010101010001000 */
    0xa488,    /* 1100010010001000 */
    0x0000,    /* 0000000000000000 */
    0x0000,    /* 0000000000000000 */
    0x0000,    /* 0000000000000000 */
    0x0000,    /* 0000000000000000 */
    0x0000,    /* 0000000000000000 */
  };
  register int i;

  linea0();                 /* initialise */
  PTSIN=pts;                /* setup ptsin */
  CONTRL=contrl;            /* and contrl */
  contrl[1]=sizeof(pts)/(sizeof(short)*2)-1;
  COLBIT0=1;                /* use all bit planes */
  COLBIT1=1;
  COLBIT2=1;
  COLBIT3=1;
  WMODE=0;                  /* replace mode */
  PATPTR=pattern;           /* set up pattern pointer */
  PATMSK=sizeof(pattern)/sizeof(short)-1;
  MFILL=0;                  /* no multi-plane fill */
  CLIP=0;                   /* no clipping */

  Y1=0;                     /* step over all lines used */
  for (i=0; i<200; i++)
  {
    linea6();               /* render one line */
    Y1++;                   /* move to next line */
  }
  return 0;
}
```

linea7 — Perform a BITBLiT

Class: Line-A — *Category: BITBLiT Functions*

SYNOPSIS

```
#include <linea.h>

linea7(blit);

LA_BLIT *blit;   pointer to blit structure
```

DESCRIPTION

linea7 is the system BITBLiT primitive (**bit block transfer**), and is unusual in that it does not use the input array. The function is passed a pointer to an LA_BLIT structure, which has the form:

```
typedef struct la_blk
{
  short bl_xmin;          minimum x
  short bl_ymin;          minimum y
  short *bl_form;         word aligned memory form
  short bl_nxwd;          offset to next word in line
  short bl_nxln;          offset to next line in plane
  short bl_nxpl;          offset to next plane
} LA_BLK;

typedef struct la_blit
{
  short bb_b_wd;          width of block in pixels
  short bb_b_ht;          height of block in pixels
  short bb_plane_ct;      number of planes
  short bb_fg_col;        foreground colour
  short bb_bg_col;        background colour
  char bb_op_tab[4];      fg/bg logic table
  struct la_blk bb_s;     source info block
  struct la_blk bb_d;     destination info block
  short *bb_p_addr;       pattern buffer address
  short bb_p_nxln;        offset to next pattern line
  short bb_p_nxpl;        offset to next pattern plane
  short bb_p_mask;        pattern index mask
  char bb_fill[24];       work space
} LA_BLIT;
```

The function performs a blit from a source to a destination form. The source form has a top left corner (bb_s.bl_xmin, bb_s.bl_ymin) with width and height bb_b_wd and bb_b_ht respectively. bb_plane_ct bit planes are then transferred to the destination form with top left corner (bb_d.bl_xmin, bb_d.bl_ymin). Note that the algorithm employed deals successfully with overlapping forms.

The remaining parameters of the source and destination form definitions (bb_s.bl_... and bb_d.bl_...) are the pointer to the base of the form bl_form, bl_nxwd, the offset to the next word in the same plane (i.e. skipping the interleaved planes), bl_nxln a count of the number of bytes in one line of the form and finally bl_nxpl, the offset to the next plane from the start of one plane, thus allowing blitting from a linear form in memory to the interleaved plane structure of the ST display.

As the planes are transferred by the blit operation, a logical operation is performed on the bits. The operations are a generalisation of those performed for the VDI vro_cpyfm routine. The logic table consist of 4 bytes, indexed by considering the value of the bits in foreground and background colours, bb_fg_col and bb_bg_col. The logic operation used for a particular bit plane is obtained by considering bb_op_tab(bb_fg_col * 2 + bb_bg_col). The logical operations are identical to those discussed under vro_cpyfm (S_AND_D etc.).

The final variant available with linea7 allows a pattern to be ANDed into the source prior to being combined with the destination. To enable the pattern integration, bb_p_addr should point to an array of patterns, similar to those used for linea4. Note that if you do not require the pattern facility you should set b_p_addr to NULL. p_nxln and p_nxpl are used identically to bl_nxln and bl_nxpl, discussed above for forms, but apply instead to the pattern 'form'. Note that p_nxln must be an exact power of two. p_mask is used with p_nxln to mask the appropriate part of the source. If p_nxlen has a value of 1<<n (i.e. an exact power of two), then the value for p_mask is (p_nxln/2-1)<<n.

The remaining 24 bytes of the LA_BLIT structure, bb_fill, are used internally by the blit algorithm.

SEE

lineae, vro_cpyfm, vrt_cpyfm

CAVEATS

This call makes almost no checks as to the validity of what is being attempted, so great care should be taken when using it as it is very easy to disrupt the machine without due care.

This function pays no regard to any clipping rectangle installed in the Line-A input array.

EXAMPLE

```
/*
 * blit the top left of the screen to the bottom
 * right
 */

#include <linea.h>
#include <vdi.h>
#include <string.h>
#include <osbind.h>
#include <stddef.h>

int main(void)
{
  LA_BLIT blt;

  linea0();

  blt.bb_b_wd=V_X_MAX/2-1;       /* blit half screen */
  blt.bb_b_ht=V_Y_MAX/2-1;
  blt.bb_plane_ct=VPLANES;       /* number of planes */
  blt.bb_fg_col=1;               /* maintain colours */
  blt.bb_bg_col=1;
  memset(blt.bb_op_tab,S_OR_D,sizeof(blt.bb_op_tab));
  blt.bb_s.bl_xmin=blt.bb_s.bl_ymin=0;
  blt.bb_s.bl_form=blt.bb_d.bl_form=Logbase();
  blt.bb_s.bl_nxwd=blt.bb_d.bl_nxwd=1<<VPLANES;
  blt.bb_s.bl_nxln=blt.bb_d.bl_nxln=VWRAP;
  blt.bb_s.bl_nxpl=blt.bb_d.bl_nxpl=2;
  blt.bb_p_addr=NULL;            /* no pattern */
  blt.bb_s.bl_xmin=blt.bb_s.bl_ymin=0;
  blt.bb_d.bl_xmin=V_X_MAX/2;
  blt.bb_d.bl_ymin=V_Y_MAX/2;

  linea7(&blt);                  /* blit it */
  return 0;
}
```

linea8 — Render bit-mapped character on screen

Class: Line-A *Category: BITBLiT Functions*

SYNOPSIS

```
#include <linea.h>

linea8();

FBASE=font;             base of font form
FWIDTH=width;           width of font form
SRCX=ch;                X co-ordinate of character in
                        font form
SRCY=0;                 Y co-ordinate of character in
                        font form
DELX=w;                 width of character
DELY=h;                 height of character
DSTX=x;                 X co-ordinate to plot character
                        on screen
DSTY=y;                 Y co-ordinate to plot character
                        on screen
TEXTFG=fgcol;           foreground colour
TEXTBG=bgcol;           background colour
STYLE=effect;           text effect
LITEMASK=lmask;         lightening mask
SKEWMASK=smask;         skewing mask
WEIGHT=thick;           thickening width
ROFF=roff;              skewing offset above baseline
LOFF=loff;              skewing offset below baseline
SCALE=enable;           enable scaling
XDDA=0x8000;            scaling variable
DDAINC=factor;          scaling factor
SCALDIR=dir;            scaling direction
CHUP=angle;             rotation angle
MONO=monoflag;          mono-spaced flag
SCRTCHP=buffer;         work buffer
SCRPT2=offset;          scaling offset into buffer
WMODE=mode;             writing mode
CLIP=state;             clipping flag.
XMINCL=x1clip;          left edge X clipping
YMINCL=y1clip;          top edge Y clipping
XMAXCL=x2clip;          right edge X clipping
YMAXCL=y2clip;          bottom edge Y clipping.
```

DESCRIPTION

linea8 is used for rendering bit-mapped fonts on screen. A character from the font at pixel offset (SRCX, SRCY) with width and height DELX and DELY is transferred to the destination (DSTX, DSTY). TEXTFG and TEXTBG give the foreground and background colours which should be used when rendering.

To enable font scaling SCALE is made non-zero, and the direction of scaling put in SCALDIR; 0 for down, otherwise up. When using scaling the fixed point variable XDDA should be initialised to 0.5 (0x8000 in the representation used), and the DDA scaling factor set up. If the final size required is final and the actual font size is actual then for scaling up DDAINC should be set to 0x100 * (final - actual)/actual, else for scaling down 0x100 * final/actual.

The effects applied to the font may be set via the STYLE bitmap:

Bit	Effect
0	**Thicken**
1	'Lighten'
2	*Skew*
3	Underline (in-operative)
4	Outline

To rotate the text CHUP may be set to the number of degrees required times 10, in the same way as vst_rotation. MONO should be set to 1 for mono-spaced fonts or non-zero for proportional fonts. SCRTCHP should be set to a word aligned scratchpad area in which text special effects are rendered. The size of this buffer should be twice the size of the largest character which may result. SCRPT2 gives an offset into the SCRTCHP buffer which is used when scaling fonts. Like the SCRTCHP buffer it should have space for twice the largest character which may result.

When rendering the text into the destination form the writing mode WMODE is used as described under linea3, however this is extended from the normal set of four modes and any of the BITBLiT modes may be used (S_AND_D etc.), by adding 4 to the normal value.

The remaining variables which must be set are normally copied directly from the font header.

An optional clipping rectangle may be specified with this function, it has top left corner (XMINCL, YMINCL) and bottom right corner (XMAXCL, YMAXCL). To enable clipping CLIP should be set to 1, or disabled by setting CLIP to 0.

SEE

linea7, v_gtext, v_justified, vst_rotation

EXAMPLE

```
/*
 * write out text in all styles, not rotated or
 * scaled
 */
#include <linea.h>
int main(void)
{
  register int i;

  linea0();
  for (i=0; i<0x20; i++)
  {
    register const char *s="Hello World";
    register char c;

    /* set up initial screen X co-ordinate */
    DSTX=0;
    if (i>=0x10)
      DSTX=V_X_MAX/2;
    while (c=*s++)
    {
      short x[500];

      TEXTFG=1;         /* colours */
      TEXTBG=0;
      STYLE=i;          /* and style */
      /* compute source position and height */
      c-=V_DEF_FONT->font_low_ade;
      SRCX=V_DEF_FONT->font_char_off[c];
      SRCY=0;
      DELX=V_DEF_FONT->font_char_off[c+1]-SRCX;
      DELY=V_DEF_FONT->font_height;
      FBASE=V_DEF_FONT->font_data;
      FWIDTH=V_DEF_FONT->font_width;
      /* copy masks and effects */
      LITEMSK=V_DEF_FONT->font_lightening;
      SKEWMSK=V_DEF_FONT->font_skewing;
      WEIGHT=V_DEF_FONT->font_thickening;
      /* offsets for skewed text */
      if (STYLE & 1<<2)    /* skewed */
      {
        ROFF=V_DEF_FONT->font_right_off;
        LOFF=V_DEF_FONT->font_left_off;
      }
      else
        ROFF=LOFF=0;
      SCALE=0;
      XDDA=0x8000;          /* initialise anyway */
      DDAINC=256;
      SCALDIR=0;
      CHUP=0;
      MONO=0;
      SCRTCHP=x;
      SCRPT2=sizeof(x)/2;
      CLIP=0;
      DSTY=(i&0xf)*V_DEF_FONT->font_height;
      /* no need to redo DSTX as linea8 does it */
      linea8();
    }
  }
}
```

linea9 — Show mouse cursor

Class: Line-A *Category: Sprite Manipulation*

SYNOPSIS

```
#include <linea.h>

linea9();

INTIN[0]=force;  zero to force mouse to show

showmouse(force);
```

DESCRIPTION

linea9 is identical to the VDI call v_show_c and is used to decrease the mouse hide depth. It takes a single parameter in INTIN(0), which if zero forces the mouse hide depth counter to be reset and the mouse displayed regardless; if it is non-zero then the hide depth is reduced by one and the mouse displayed if the hide depth becomes zero.

The showmouse macro is provided to simplify the interface and takes a single parameter force, as described above.

SEE

lineaa, v_show_c, v_dspcur, graf_mouse

lineaa

Hide mouse cursor

Class: Line-A *Category: Sprite Manipulation*

SYNOPSIS

```
#include <linea.h>

lineaa();

hidemouse();
```

DESCRIPTION

lineaa (and the equivalent name hidemouse) is identical to the VDI call v_hide_c and is used to increase the mouse hide depth. When the hide depth is non-zero the mouse cursor is not displayed.

SEE

linea9, v_hide_c, v_rmcur, graf_mouse

lineab	Transform mouse cursor

Class: Line-A *Category: Sprite Manipulation*

SYNOPSIS

```
#include <linea.h>

lineab();
```

DESCRIPTION

lineab is identical is used to change the form of the mouse cursor, in an identical manner to vsc_form. A pointer to an LA_SPRITE structure is placed in INTIN(0-1) giving the new form:

```
typedef struct la_sprite
{
  short ls_xhot;         X hot spot offset
  short ls_yhot;         Y hot spot offset
  short ls_form;         1 for VDI, -1 for XOR
  short ls_bgcol;        background colour index
  short ls_fgcol;        foreground colour index
  short ls_image[32];    interleaved image
} LA_SPRITE;
```

The image is stored in image/mask interleaved form. The first word in the ls_image array gives the mask, the second the data, the third the mask, etc. The ls_form value gives the way the mouse is rendered on screen. For both VDI and XOR modes, most combinations are identical:

Foreground	Background	Colour plotted
0	0	Destination
0	1	Background
1	0	Foreground (VDI mode) Inverse destination (XOR mode)
1	1	Foreground

To save the old mouse form before changing it the old form should be copied from V_MASK_FORM (note that the full LA_SPRITE structure for the mouse cursor starts at V_M_POS_HX). Also when changing the mouse form you should disable drawing of the mouse by setting V_MOUSE_FLAG to 0 and restore it afterwards. This ensures that 'droppings' do not occur.

SEE

linead, vsc_form, graf_mouse

lineac

Remove user sprite

Class: Line-A *Category: Sprite Manipulation*

SYNOPSIS

```
#include <linea.h>

lineac(save);

void *save;   pointer to sprite save area
```

DESCRIPTION

lineac is used to remove a sprite previously drawn using linead. A pointer to the sprite save area is passed and the screen restored from this.

SEE

linead, linea9, lineaa

linead

Render user sprite

Class: Line-A *Category: Sprite Manipulation*

SYNOPSIS

```
#include <linea.h>

linead(x,y,sprite,save);

int x;                     x position for sprite
int y;                     y position for sprite
LA_SPRITE *sprite;         pointer to sprite definition
void *save;                pointer to sprite save area
```

DESCRIPTION

linead is used to render a user defined sprite. The position to plot the sprite at is passed in x and y, and the sprite definition in sprite. sprite is a pointer to an LA_SPRITE structure, discussed previously under lineab.

The save area is used to keep a copy of the screen area corrupted by the sprite. It shares the first 5 fields of the LA_SPRITE structure, but must have room for the image from all screen bitplanes. Hence it should have a size of 10+VPLANES*64 bytes.

SEE

lineac, lineab

lineae

Copy raster form

Class: Line-A *Category: BITBLiT Functions*

SYNOPSIS

```
#include <linea.h>

lineae();

INTIN[0]=wr_mode;    logic operation to perform
CONTRL[7-8]=src;     source memory form definition
                     block
CONTRL[9-10]=dest;   destination memory form
                     definition block
INTIN[1]=one_col;    colour index for 1s in the data
INTIN[2]=zer_col;    colour index for 0s in the data
PTSIN[0]=llx1;       lower-left X of first rectangle
PTSIN[1]=lly1;       lower-left Y of first rectangle
PTSIN[2]=urx1;       upper-right X of first rectangle
PTSIN[3]=ury1;       upper-right Y of first rectangle
PTSIN[4]=llx2;       lower-left X of second rectangle
PTSIN[5]=lly2;       lower-left Y of second rectangle
PTSIN[6]=urx2;       upper-right X of second rectangle
PTSIN[7]=ury2;       upper-right Y of second rectangle
COPYTRAN=mode;       opaque/transparent mode
CLIP=state;          clipping flag.
XMINCL=x1clip;       left edge X clipping
YMINCL=y1clip;       top edge Y clipping
XMAXCL=x2clip;       right edge X clipping
YMAXCL=y2clip;       bottom edge Y clipping.
```

DESCRIPTION

lineae is the VDI raster copy primitive and performs the equivalent of both vrt_cpyfm and vro_cpyfm. Referring to the description of vrt_cpyfm and vro_cpyfm, the parameters discussed there are placed in the arrays as noted above. To perform a vro_cpyfm, COPYTRAN should be set to 0, or 1 to perform a vrt_cpyfm.

When COPYTRAN is 1, one_col and zer_col should be provided to give the colours for ones and zeroes respectively, as discussed under vrt_cpyfm.

An optional clipping rectangle may be specified with this function; it has top left corner (XMINCL, YMINCL) and bottom right corner (XMAXCL, YMAXCL). To enable clipping CLIP should be set to 1, or disabled by setting CLIP to 0.

SEE

linea7, vro_cpyfm, vrt_cpyfm

lineaf	Flood fill area

Class: Line-A *Category: Area Filling*

SYNOPSIS

```
#include <linea.h>

lineaf();

INTIN[0]=colour;      colour to search for
PTSIN[0]=x;           x co-ordinate of start point
PTSIN[1]=y;           y co-ordinate of start point
WMODE=mode;           writing mode
PATPTR=pattern;       pointer to fill pattern
PATMSK=index;         pattern count
MFILL=flag;           multi plane fill flag
CLIP=state;           clipping flag.
XMINCL=x1clip;        left edge X clipping
YMINCL=y1clip;        top edge Y clipping
XMAXCL=x2clip;        right edge X clipping
YMAXCL=y2clip;        bottom edge Y clipping
SEEDABORT=fn;         abort fill pointer
```

DESCRIPTION

lineaf is used to flood fill an area (often called seed fill), in an identical manner to v_contourfill. The x and y parameters of v_countourfill are passed in PTSIN(0) and PTSIN(0), whilst the boundary colour is passed in INTIN(0).

The PATPTR, PATMSK, MFILL and WMODE parameters are as described under linea4. Note that the COLBIT values are not passed to this function, instead the current workstation fill colour attribute is used, hence this function must always be used with an open workstation.

A clipping rectangle *must* be specified with this function, it has top left corner (XMINCL, YMINCL) and bottom right corner (XMAXCL, YMAXCL). Note that the clipping flag CLIP is ignored, clipping is always performed.

SEEDABORT is a function called after plotting every line, and is used to abort the fill. If the function called returns 0 the flood fill continues, otherwise it is aborted.

SEE

linea4, v_countourfill

CAVEATS

This function does not evaluate the COLBIT values for its drawing colour and the colour used is that of the current workstation, hence a workstation must be opened by the application. The function is still however of great use as it allows an abort function to be specified so that a user may abort an incorrect or 'leaking' fill.

EXAMPLE

```
/*
 * draw a circle on screen and then seed-fill it
 */
#include <linea.h>
#include <vdi.h>
#include <aes.h>

short __saveds sab(void)
{
  return V_MOUSE_BT;  /* stop when a button pressed */
}

int main(void)
{
  short v_handle,junk;
  short pattern[]={
    0x0940,    /* 0000100101000000 */
    0x0f40,    /* 0000111101000000 */
    0x0940,    /* 0000100101000000 */
    0x64dc,    /* 0110010011011100 */
    0x8a88,    /* 1000101010001000 */
    0xcac8,    /* 1100101011001000 */
    0x2a88,    /* 0010101010001000 */
    0xa488,    /* 1100010010001000 */
  };

  appl_init();
  v_handle=graf_handle(&junk,&junk,&junk,&junk);
  linea0();
  hidemouse();
  vs_clip(v_handle,0,NULL);
  vswr_mode(v_handle,MD_REPLACE);
  vsf_color(v_handle,BLACK);
  vsf_interior(v_handle,FIS_HOLLOW);
  vsf_perimeter(v_handle,1);
  v_circle(v_handle,V_X_MAX/2,V_Y_MAX/2,V_Y_MAX/2);
  PTSIN[0]=V_X_MAX/2;
  PTSIN[1]=V_Y_MAX/2;
  INTIN[0]=-1;
  XMINCL=YMINCL=0;
  XMAXCL=V_X_MAX;
  YMAXCL=V_Y_MAX;
  SEEDABORT=sab;
  WMODE=MD_REPLACE;
  PATPTR=pattern;
  PATMSK=sizeof(pattern)/sizeof(short)-1;
  MFILL=0;
  lineaf();
  showmouse(1);
  return appl_exit();
}
```

Index