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.