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_multi.

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 an 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 an 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;
    }
```