ATARI.RSC
The Atari Developer's Resource

Vol. VI, Issue 1
April 1993

No Program Is An Island

Mike Fulton

Ask yourself this question: "My application does (fill-in-the-blank). Have I looked at programs on other systems that do that, like Program-X for Microsoft Windows? Or Program-Y for the Macintosh?"

The answer is quite often that the developer hasn't compared their software against software for other platforms. You may ask why looking at other systems is important, since that's not the competition for an Atari-based package. The answer is that the competition does include software for other platforms like the Macintosh and Microsoft Windows. And it's always a good idea to keep track of what the competition is doing. It's one of the most basic rules of doing business.

Among existing Atari users, other Atari-based programs are going to be your main competition. But you must also consider people who are looking to buy their first computer. If an Atari user decides not to buy your software now, there's always a chance they may change their mind later. But once somebody has bought a different computer, you've probably lost that sale forever.

Statistics show that most software, except games, is sold to users during the first year they own their computer. When a person buys a computer, their decision is usually based on the ability to use a particular type of program. That means that new users will be comparing your software against software for other platforms <u>before</u> they decide what computer to purchase. So you have to be aware of what other programs can do, even if they are for different computer systems. What can you learn from looking at them? What features are popular or not, and why?

It doesn't matter if you don't like MS Windows and MSDOS-based computers. Your opinion of the Macintosh or Amiga doesn't matter. Whatever your feelings are about other platforms, you have potential customers who may not agree with you.

It would be great if you had a PC running MS Windows, or maybe a Macintosh or an Amiga. Then you could buy programs for these systems to check them out. But as useful as this idea can be, it is not always financially practical, or possible, for a developer to do this. But that doesn't mean you can't keep track of what's going on.

If you don't already subscribe to a few PC and Macintosh magazines, you should. These magazines have advertising, in-depth reviews, and comparisons of all the latest products for those platforms. If you're creating a new word processor for the Atari, what do you think you can learn by looking at a comparison of 10 different word processors for MS Windows?

Visit your local software stores and look at the packaging for other products. What features do they make a big deal about? Depending on the product and the store, you may even be able to get demonstrations of different software packages.

Imagine your program on a different platform. Would it be competitive with the other programs in the same category? Maybe your program got a good review in one of the Atari magazines, but what kind of review would it get in one of the PC or Macintosh magazines?

Another question to ask yourself is: Are you are getting the most out of the GEM user interface? Are there things you could be doing in your program's user interface to make it more accessible and easier to use? Is there something that Program-X is doing with MS Windows that you could be doing with GEM? If not, then why not?

Some developers on the Atari platform are doing these things already. And because of it, their software is higher quality, easier to use, has better documentation, and is more successful in the marketplace. Even if you forget about other platforms for a minute, remember that you still have to compete with these Atari developers. And as the Falcon030's market grows, the competition will get more and more intense.

One of Atari's mottos has been "Power without the Price", so maybe you're aiming your program at a lower price point and think that it's OK to have a limited feature set and less sophisticated user interface. But consider that there are programs at lower price points on the PC and Macintosh that are very competitive with more expensive packages. Less expensive doesn't mean disabled.

(continued on page 2)

Inside This Issue

ATARI.RSC
The Resource File

DIRECTOR of APPLICATION SOFTWARE
Bill Rehbock (408) 745-2082
Fax: (408) 745-2088
GEnie: B.REHBOCK
CIS: 75300,1606
Internet:
75300.1606@compuserve.com

DEVELOPER TECHNICAL SUPPORT
J. Patton (408) 745-2135
GEnie: ATARIDEV
CIS: 70007,1072
Internet:
70007.1072@compuserve.com

Mike Fulton (408) 745-8821
GEnie: MIKE-FULTON
CIS: 75300,1141
Internet:
75300.1141@compuserve.com

Fax: (408) 745-2088

ATARI.RSC edited by Mike Fulton

ATARI.RSC was created using Pagestream v2.2 on the Atari TT030 computer & Atari TTM194 monochrome monitor, and was printed using CompoScript on the Atari SLM605 Laser Printer.

No Program Is An Island
(continued from page 1)

With the introduction of the new Atari Falcon030, there will soon be a lot of new users in the Atari market. And a lot of new developers have been signed up to work on products for the Falcon030. Many of these developers are new to the Atari platform and they are bringing a lot of experience from other platforms with them, including a pretty good idea of what to expect from new users. These new users aren't going to be judging your products by the same standards as the existing Atari userbase. They will probably be much harder on you than that. If they see a product on another machine doing something your product can't do, they will be wanting to know why not. What are you going to tell them?

This article isn't aimed at any one developer in particular, but if you think it's talking about you, then you're probably right. The question is, what are you going to do about it?

Correction

The* ATARI ST/TT Q&A *column from the last issue had a typographical error in one of the questions. Here's the corrected version:

Q: How do I have my program look for an event on either mouse button using ***evnt_multi()*** or ***evnt_button()***?

A: Use an *ev_mbclicks* value of 0x101, a *ev_mbstate* value of 0, and an *ev_mbmask* value of 3 with the ***evnt_multi()*** and ***evnt_button()*** functions and you will get an event whenever either mouse button is pressed. This method has worked in all versions of TOS and is now official. (This method does not work for detecting mouse button releases.)

Documentation Correction

On page VIDEO.6 of the Falcon030 developer's documentation, the description of the ***VsetMask()*** call should be changed to read as follows:

> *OPCODE 150*
> ***VsetMask(*** *ormask, andmask, overlay)*
> *LONG ormask, andmask;*
> *WORD overlay;*

The ***VsetMask()*** function is used to set the mask values used by VDI to modify the color values computed for ***vs_color()***. The ***vs_color()*** function converts its input to a 16-bit RGB value which is bitwise OR'ed with *ormask* and then bitwise AND'ed with *andmask*. This allows the application to set any color to be transparent (or not) in the 15-bit per pixel true color modes with genlock and overlay.

The default mask values are: *ormask* = 0x00000000, *andmask* = 0xFFFFFFFF. This combination of mask values has no effect.

To set the overlay bit, use *ormask* = 0x00000020, *andmask* = 0xFFFFFFFF. Now any colors set with ***vs_color()*** will have the overlay bit set.

To clear the overlay bit, use *ormask* = 0x00000000, *andmask* = 0xFFFFFFDF. Now any colors set with ***vs_color()*** will have the overlay bit cleared.

If the *overlay* value is non-zero, then the system will be put into overlay mode. If the *overlay* value is zero, then the system will be taken out of overlay mode.

Popup Menus & Hierarchical Submenus, Part 1

Mike Fulton

The new GEM AES versions found in MultiTOS (v4.0) and the Falcon030 (v.3.3) now support a new class of features for adding popup menus and hierarchical submenus to your user interface design.

You've probably seen popup menus in at least a couple of programs. For example, the Resource Construction Set included in the Atari Developer Kit uses popup menus for changing object state and flag values, as well as object colors and fill patterns.. However, before now if you wanted to have popup menus in your program, you had to program all the necessary routines yourself from scratch. Now you can do it much more easily, using the new ***menu_popup()*** function which is now available in the new GEM AES.

Much less common on the Atari platform than popup menus are programs with hierarchical menus. This is because without built-in support, they are much more difficult to do than popup menus, especially for submenus linked to menu entries in the main menu bar. They should become much more common now that the new AES hierarchical menu features allow you to easily add submenus to either an item popup menu or even to a menu item inside the main menu bar. You can even put as many items in to a submenu as you like, and the system will automatically scroll through them as necessary.

Part 1 of this article will show you how to add a pre-fabricated popup menu to a dialog box, and how to link a hierarchical submenu to one of the popup menu items. Part 2 of this article, presented in the next issue of ATARI.RSC, will show you how to build a submenu at runtime and link it to a menu item in your program's main menu bar.

The sample program, resource file, and complete source code are available in the ATARI.RSC Roundtable on GEnie and Library 7 of the ATARIPRO forum on Compuserve. The source code is presented in listings #1-5 at the end of the article so that you may follow along in the source code as you read.

Note: This program is written simply as an example of how to use these new features, and does not necessarily reflect an ideal design. You must have a Falcon030 or have MultiTOS installed in order to see the popup menus and hierarchical submenus in this program.

This article is not intended to serve as primary documentation for all of the new functions. For full documentation on the new menu library functions, see one of the following:

Falcon030 Developer Documentation: AES section

GEM AES Programmer's Guide, Revision 4
(avail: 2nd Qtr. 93)

GEM.DOC from MultiTOS Developer Distribution
(available online)

Program Outline

The program is very simple. In Listing #1, the ***main()*** function registers the program with GEM AES with the ***appl_init()*** function. After this function, the GEM AES version number is available in one of the AES parameter arrays, so the program stores this value for later use.

After initialization, the ***sample_dialog()*** routine is called. This function contains the main part of the example program and is responsible for showing the sample dialog box shown in figure #1.

Many developers will be concerned about having their programs work on machines with older AES versions as well as taking advantage of the new features of the newer AES versions. Therefore, the example program is designed in part to take this into account and use popup menus and submenus if those features are available, and to do things another way if they aren't.

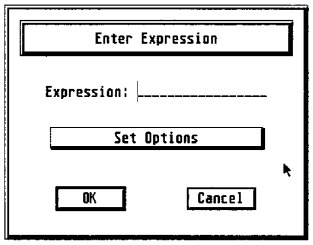

figure #1

The ***sample_dialog()*** routine is set up so that if a proper version of GEM AES is running, it will use the popup menu and hierarchical menu functions to display a submenu that will appear when the *Set Options* button is clicked on, as shown in figure #2.

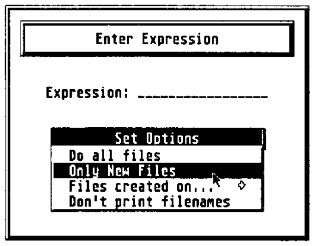

figure #2

If an older version of GEM AES is running, then when the *Set Options* button is selected, a second dialog will appear to show most of the same options that would have appeared in the popup menu (all except the hierarchical submenu options we'll see later). ***NOTE: This is just an example and by no means should be considered the best method of handling this situation. In your own programs you may want to do something else, possibly even including requiring the new AES in order to run at all.***

New Functions & Data Structures

The menu library has four new features altogether, along with a couple of new data structures. See the new menu library documentation for complete information on the new functions and data structures.

In our example program, we will be using two of the new menu library functions, ***menu_attach()*** and ***menu_popup()***, along with the ***MENU*** structure. The ***MENU*** structure can be used for both input and output, depending on the function and parameter being used.

```
typedef struct
{
   OBJECT *mn_tree;
   WORD mn_menu;
   WORD mn_item;
   WORD mn_scroll;
   WORD mn_keystate;
} MENU;
```

menu_attach()

```
WORD menu_attach( me_flag, me_tree, me_item, me_mdata );
WORD me_flag;
OBJECT *me_tree;
WORD me_item;
MENU *me_mdata;
```

In our example program, we're going to have a hierarchical submenu linked to one of the items in our popup menu, as shown in figure #3. By using the ***menu_attach()*** function, we can have GEM AES manage this for us automatically. In our example program, we call ***menu_attach()*** about 25 lines into the ***sample_dialog()*** function:

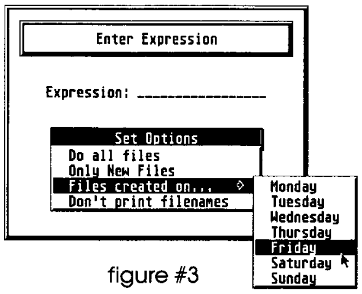

figure #3

```
sub_attached = menu_attach( 1, pop1.mn_tree, SUB_OPT_H,
                            &days );
```

The *me_flag* parameter indicates what we're doing. We can either link a submenu to a menu item (*me_flag* = 1) or unlink one (*me_flag* = 2), or we can ask the system what submenu is already linked to a particular menu item (*me_flag* = 0). At the start of ***sample_dialog()***, we're linking a submenu to a popup menu item so we use a value of 1. At the end of the function, we unlink the submenu using ***menu_attach()*** again with an *me_flag* value of 2.

The *me_tree* parameter is the address of the object tree of the menu item to which we're attaching the submenu. In our example program, we're linking it to the popup menu, so we use the address of the popup menu's object tree, contained in *pop1.mn_tree.*

Our example program links a submenu to a popup menu item. Note that if we were linking a submenu to an item in a main menu bar, as shown in figure #4, then we would instead use the address of the menu bar object tree. We would then be told of user selections in the submenu through the MN_SELECTED message, which

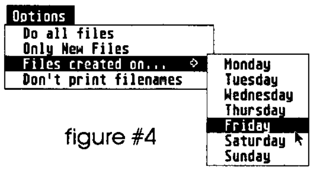

figure #4

has been extended to support hierarchical submenus. (See the main menu library documentation.)

The *me_item* parameter is the object index of the menu item within the *mn_tree* object tree that we wish to attach our submenu to. We want it linked to the 3rd object in the popup menu, which is defined in our resource include file as *SUB_OPT_H.*

The *me_mdata* parameter is a pointer to a MENU structure containing information about the submenu. In this example, that's the *days* structure, used as follows:

days.mn_tree = object tree address of submenu

days.mn_menu = parent object of submenu items. Note that it's possible for one object tree to contain several different submenus or popup menus by placing each one in its own parent box object. In our example, each menu is a stand-alone object tree, so we've set this value to the ROOT object (defined in the C header files as 0).

days.mn_item = object index of 1st submenu item to be shown. For menus that show several options, of which you can only have one selected at a time, a font selector for example, this should be set to the object matching the current selection. In our example, you can select any or all of the items, so we always start at the top item.

days.mn_scroll = This a flag that indicates if we want to scroll this menu or not. If you set this to zero, your menu will not scroll and can contain any valid object type, including user-defined objects. If you set the menu to scroll if need be, however, it must contain only G_STRING objects. We don't care if our menu scrolls, so we set this to 1.

days.mn_keystate = unused here

menu_popup()

```
WORD menu_popup( menu, xpos, ypos, mdata )
MENU *menu;
WORD xpos, ypos;
MENU *mdata;
```

The ***menu_popup()*** function is used to display a popup menu and get back the user's selection. Our example program initializes the MENU data structure near the beginning of the ***sample_dialog()*** function.

The *menu* parameter is a pointer to a MENU structure that contains information about the popup menu you wish to display. In our example program, that's the *pop1* structure, used as follows:

pop1.mn_tree = object tree address of popup menu

pop1.mn_menu = parent object within tree for submenu items. Since our popup menu is a stand-alone object tree, the parent is the ROOT object.

pop1.mn_item = Starting menu item, if scrolling menu. We set this to the first item in the submenu.

pop1.mn_scroll = Flag, can we scroll this menu? We use a value of 1, indicating it can be scrolled.

pop1.mn_keystate = unused here

The *xpos* and *ypos* parameters are simply the screen coordinates at which to display the popup menu. You get the best performance if the *xpos* value is byte aligned. However, in our example program, we simply get the screen location of the button that leads to the popup menu, and position the popup right underneath.

The *mdata* parameter is a pointer to a MENU structure that will get the results of the user's selection from the popup menu. In our example program, that's the *pop2* structure. After we call ***menu_popup(),*** the values contained in this structure are:

pop2.mn_tree = object tree address of item selected. Note that if you have hierarchical submenus attached to popup menu items, this value will not necessarily be the address of the popup menu itself, but could be the address of the hierarchical submenu's tree.

pop2.mn_menu = parent object index of selected item.

pop2.mn_item = Object index of item selected

pop2.mn_scroll = unused here

pop2.mn_keystate = the status of the Control, Shift, and Alternate keys when the mouse button was pressed.

Once we determine that the appropriate button has been pressed, we highlight the button and then we do the ***menu_popup()*** call as detailed above. After this, the button is unhighlighted. If no error was returned from ***menu_popup(),*** then the we get the object tree and object index from the ***pop2*** structure and toggle a checkmark on the selected item.

Once we've finished showing the popup, we loop back into the form_do() function to await the next user action in the dialog. When the user selects either the *OK* or the *CANCEL* button, we release the screen area belonging to the dialog and release ***wind_update().***

The Program Listings

The program is set up to be compiled with either Lattice C v5 or Mark Williams C, but with a little work should be portable to other compilers.

Since the GEM libraries for these compilers don't include bindings for the new AES functions, the files AESBIND.C, AESBINDR.H, and AESCALL.H are included. These files contain bindings for the new ***menu_attach()*** and ***menu_popup()*** calls. The AESBINDR.H and AESCALL.H files include define statements and external declarations to allow the same bindings to work with either compiler.

The AESCALL.H file includes a bit of inline code for making the TRAP #2 call to GEM AES. It is separated into its own file because Mark Williams C would not work with the #pragma statement. This way it is included only when using Lattice C.

Listing #1 - POPUP.C

```
#include "aesbindr.h"
#include "menu.h"
#include "popup.h"

short gl_apid, aes_version, xdial, ydial, hdial, wdial;
OBJECT *obj_addr;
char rsrcname[] = "POPUP.RSC";
char prgname[] = "  Popup Menu Example ";

void multi(), sample_dialog();

/*************************************************************************/

void main()
{
extern short _AESglobal[];

        gl_apid = appl_init();
        aes_version = _AESglobal[0];    /* Save GEM AES version number */

        if( rsrc_load( rsrcname ) )
        {
                graf_mouse( ARROW, 0L );
                multi();
                rsrc_free();
        }
        appl_exit();
}

void multi()
{
        sample_dialog();
}

void set_options()
{
short exit_obj;

        rsrc_gaddr( R_TREE, OPTIONS, &obj_addr );
        graf_mouse( ARROW, 0L );

/* Display the options dialog box */

        wind_update( BEG_UPDATE );
        form_center( obj_addr, &xdial, &ydial, &wdial, &hdial );
        form_dial( 0, 0, 0, 0, 0, xdial, ydial, wdial, hdial );
        form_dial( 1, 0, 0, 0, 0, xdial, ydial, wdial, hdial );
        objc_draw( obj_addr, ROOT, MAX_DEPTH, xdial, ydial, wdial, hdial );
        exit_obj = form_do( obj_addr, 0L );
        form_dial( 2, 0, 0, 0, 0, xdial, ydial, wdial, hdial );
        form_dial( 3, 0, 0, 0, 0, xdial, ydial, wdial, hdial );
        wind_update( END_UPDATE );

        obj_addr[exit_obj].ob_state &= ~SELECTED;
}

void sample_dialog()
{
OBJECT *tmptree;
short exit_obj, pop_x, pop_y, no_err, sub_attached, checkmark;
char *x;
MENU pop1, pop2, days;

        rsrc_gaddr( R_TREE, EXPRESSION, &obj_addr );

/* If we're running on new AES, set up the popup menu. */

        if( aes_version >= 0x0330 )
        {
                rsrc_gaddr( R_TREE, OPTIONS_POPUP, &pop1.mn_tree );
                pop1.mn_menu = ROOT;            /* Parent object of menu items */
                pop1.mn_item = SUB_OPT_C;       /* initial top menu item */
                pop1.mn_scroll = 1;             /* Scroll if necessary */

/* Make sure button leading to popup menu is the way we want it. */

                obj_addr[EXEC_OPTS].ob_state |= SHADOWED;
                obj_addr[EXEC_OPTS].ob_flags |= TOUCHEXIT;
                obj_addr[EXEC_OPTS].ob_flags &= ~EXIT;
                obj_addr[EXEC_OPTS].ob_flags &= ~SELECTABLE;

/* One of the popup menu choices leads to a submenu, so attach it */

                rsrc_gaddr( R_TREE, DAYS_POPUP, &days.mn_tree );
```

```
           days.mn_menu = ROOT;
           days.mn_item = DAY_1;
           days.mn_scroll = 1;
           sub_attached = menu_attach( 1, pop1.mn_tree, SUB_OPT_H, &days );
        }

/* If old AES version, then fix button leading to 2nd dialog */

        else
        {
           obj_addr[EXEC_OPTS].ob_state &= ~SHADOWED;
           obj_addr[EXEC_OPTS].ob_flags &= ~TOUCHEXIT;
           obj_addr[EXEC_OPTS].ob_flags |= EXIT;
           obj_addr[EXEC_OPTS].ob_flags |= SELECTABLE;
        }

/* Clear out editable field */

        x = ((TEDINFO *)(obj_addr[EXEC_EXP1].ob_spec))->te_ptext;
        x[0] = 0;

/* Lock down screen, reserve screen area, draw dialog's object tree */

        wind_update( BEG_UPDATE );
        form_center( obj_addr, &xdial, &ydial, &wdial, &hdial );
        form_dial( 0, 0, 0, 0, 0, xdial, ydial, wdial, hdial );
        form_dial( 1, 0, 0, 0, 0, xdial, ydial, wdial, hdial );
        objc_draw( obj_addr, ROOT, MAX_DEPTH, xdial, ydial, wdial, hdial );

/* Loop until we're ready to exit! */

        do
        {
           graf_mouse( ARROW, 0L );
           exit_obj = form_do( obj_addr, EXEC_EXP1 );

           if( exit_obj == EXEC_OPTS )
           {
/* Do the new AES popup calls only if running on new AES */

              if( aes_version >= 0x0330 )
              {
/* Position the popup just below button that leads to it. */

                 objc_offset( obj_addr, EXEC_OPTS, &pop_x, &pop_y );
                 pop_y += obj_addr[EXEC_OPTS].ob_height + 2;

/* Select the button that leads to the popup menu */

                 objc_change( obj_addr, EXEC_OPTS, 0,
                    xdial, ydial, wdial, hdial,
                    (obj_addr[EXEC_OPTS].ob_state | SELECTED), 1 );

/* Display the popup and get back the user's selection */

                 no_err = menu_popup( &pop1, pop_x, pop_y, &pop2 );

/* Deselect the button that leads to the popup menu */

                 objc_change( obj_addr, EXEC_OPTS, 0,
                    xdial, ydial, wdial, hdial,
                    (obj_addr[EXEC_OPTS].ob_state & ~SELECTED), 1 );

/* We just want to toggle the checkmark on the item the user selected... */

                 if( no_err )
                 {
                    checkmark = (pop2.mn_tree[pop2.mn_item].ob_state
                            & CHECKED) ? 1 : 0;

                    menu_icheck( pop2.mn_tree, pop2.mn_item, (! checkmark) );
                 }
              }

/* If not new AES, then get the user's selection with a second dialog */

              else
              {
                 tmptree = obj_addr;
                 set_options();         /* Call the other dialog routine */
                 obj_addr = tmptree;

/* redo the stuff that set_options() changed */

                 wind_update( BEG_UPDATE );
                 form_center( obj_addr, &xdial, &ydial, &wdial, &hdial );
                 obj_addr[EXEC_OPTS].ob_state &= ~SELECTED;
                 objc_draw( obj_addr, ROOT, MAX_DEPTH,
                          xdial, ydial, wdial, hdial );
              }
           }

/* Stay in dialog until one of the real exit button selected */

        }
        while( exit_obj != EXEC_OK && exit_obj != EXEC_CANCEL );

/* release screen area and wind update */

        form_dial( 2, 0, 0, 0, 0, xdial, ydial, wdial, hdial );
        form_dial( 3, 0, 0, 0, 0, xdial, ydial, wdial, hdial );
        wind_update( END_UPDATE );

/* Deselect exit button */

        obj_addr[exit_obj].ob_state &= ~SELECTED;
}
```

Listing #2 - MENU.H

```
typedef struct {
        OBJECT   *mn_tree;
        short mn_menu;
        short mn_item;
        short mn_scroll;
        short mn_keystate;
} MENU;

typedef struct {
        long  display;
        long  drag;
        long  delay;
        long  speed;
        short height;
} MN_SET;
```

Listing #3 - AESBINDR.H

```
#ifdef LATTICE
#include <aes.h>
#include "aescall.h"
#define VOID void
#endif

/***********************************************************************/

#ifdef MWC

#include <obdefs.h>
#include <gemdefs.h>

extern short
        global[], control[], int_in[], int_out[], addr_in[], addr_out[];

#define _AESglobal global
#define _AEScontrol control
#define _AESintin int_in
#define _AESintout int_out
#define _AESaddrin addr_in
#define _AESaddrout addr_out

#define aes(a)   crystal(a)
#define VOID char

#endif
```

Listing #4 - AESCALL.H

```
/* Do this separately because other compilers choke on #pragma statement */

/*      move.l   (sp),d1       Do this in separate file because MWC & Alcyon C
        move.l   #$c8,d0       will choke on the #pragma statemene below.
        trap  #2              They don't need it anyway...
*/

#pragma inline d0=aes() { register d0,d1,d2,a0,a1,a2; "2217203c000000c84e42"; }
```

Listing #5 - AESBIND.C

```
#include "aesbindr.h"
#include "menu.h"

/*********************************************************************/

#ifdef MWC
short *_AESpb[] = { global, control, int_in, int_out, addr_in, addr_out };
#endif

/*********************************************************************/

short
menu_popup( me_menu, me_xpos, me_ypos, me_mdata )
MENU *me_menu, *me_mdata;
short me_xpos, me_ypos;
{
        _AEScontrol[0] = 36;
        _AEScontrol[1] = 2;
        _AEScontrol[2] = 1;
        _AEScontrol[3] = 2;
        _AEScontrol[4] = 0;
        _AESintin[0] = me_xpos;
        _AESintin[1] = me_ypos;
        _AESaddrin[0] = (VOID *)me_menu;
        _AESaddrin[1] = (VOID *)me_mdata;

        aes( _AESpb );
        return( _AESintout[0] );
}


short
menu_attach( me_flag, me_tree, me_item, me_mdata )
short me_flag;
OBJECT *me_tree;
short me_item;
MENU *me_mdata;
{
        _AEScontrol[0] = 37;
        _AEScontrol[1] = 2;
        _AEScontrol[2] = 1;
        _AEScontrol[3] = 2;
        _AEScontrol[4] = 0;
        _AESintin[0] = me_flag;
        _AESintin[1] = me_item;

        _AESaddrin[0] = (VOID *)me_tree;
        _AESaddrin[1] = (VOID *)me_mdata;

        aes( _AESpb );
        return( _AESintout[0] );
```

New Cookies for the Cookie Jar

Mike Fulton

There are two new cookies for the Cookie Jar in MultiTOS and Falcon030. They are the **_IDT** cookie and the **_AKP** cookie and they are installed with any system that has GEM AES v3.3 and up.

_IDT Cookie

The **_IDT** Cookie contains user preference settings for International Date & Time formatting and can be used by a program to aid in customizing itself for different languages and countries.

For instance, in the USA we normally shorten dates into Month, Day, Year. So for March 27, 1993. we would use "3/27/93". However in Germany, this same date would be written as "27/3/93". Other countries use other variations.

Some countries use a 12 hour clock, some use a 24-hour clock, and in some countries you might find either type, depending on a particular person's preferences.

The value of the _IDT cookie specifies the user's preferences for how dates and times should be formatted. The high word is currently unused and is reserved. The low word is broken up into three sections.

```
Bits 15-12  =  Time
               0 means a 12-hour clock
               1 means a 24-hour clock

Bits 11-8  =   Date
               0 means MM-DD-YY
               1 means DD-MM-YY
               2 means YY-MM-DD
               3 means YY-DD-MM

Bits 7-0 =     Date Separator Character
               ASCII Value (a value of zero is
               equivalent to a '/' character)
```

The _AKP Cookie

The **_AKP** Cookie (Atari Keyboard Preferences) contains the user's settings for keyboard layout and language preferences.

The high word of the _AKP cookie value is currently unused and reserved for future expansion. The low word contains the user's language preferences in the high byte, and the user's keyboard layout in the low byte. These values are the same as the TOS country code, as specified below. NOTE: the entries for Czechoslovakia and Hungary are new.

```
#define USA    0       /* USA */
#define FRG    1       /* Germany */
#define FRA    2       /* France */
#define UK     3       /* United Kingdom */
#define SPA    4       /* Spain */
#define ITA    5       /* Italy */
#define SWE    6       /* Sweden */
#define SWF    7       /* Switzerland (French) */
#define SWG    8       /* Switzerland (German) */
#define TUR    9       /* Turkey */
#define FIN    10      /* Finland */
#define NOR    11      /* Norway */
#define DEN    12      /* Denmark */
#define SAU    13      /* Saudi Arabia */
#define HOL    14      /* Holland */
#define CZE    15      /* Czechoslovakia */
#define HUN    16      /* Hungary */
```

Three-D Objects with GEM AES

by Mike Fulton

It's likely that the first new feature you will notice when you use MultiTOS or a Falcon030 is that windows and dialogs for the GEM Desktop now have a sculpted 3D appearance.

Among other new features, GEM AES versions later than v3.3 (TOS 4.xx in Falcon030 or MultiTOS on any machine) recognize three new kinds of 3D effects that can be applied to existing object types. When these effects are enabled, objects will be drawn with a sculpted 3D appearance.

Figure #1 shows how the same exact dialog appears differently under TOS 3.06 and under MultiTOS using 3D objects. You may notice that there is no difference betwen the objects labeled "3D Indicator" and "3D Activator". That's because they aren't selected. When the SELECTED bit of the *ob_state* field is set, they are drawn differently, as shown in figure #2. The objects in the left column are unselected, while the ones on the right all have the SELECTED bit set. The other figures demonstrate how different objects appear with the 3D flags set. There are more possibilities than we have room to show here, but you should get the idea.

If a 3D object uses color 0 with a hollow fill pattern, then it will be drawn using the system 3D object color settings, except when running in video modes with fewer than 16 colors. Under older versions of TOS. these objects will be drawn without the special 3D effects or colors, meaning you can use the same resource in both cases.

The 3D effects are controlled by bits 9 & 10 of the *ob_flags* field of the OBJECT structure for the object. If both bit 9 and bit 10 are zero, then the object will not be drawn using any 3D attributes.

If bit 9 only of the *ob_flags* field is set, then the object is a 3D INDICATOR. This is typically used in a dialog box to indicate some sort of state, such as whether an option is "on" or "off". For example, radio buttons in a dialog box should always be indicators.

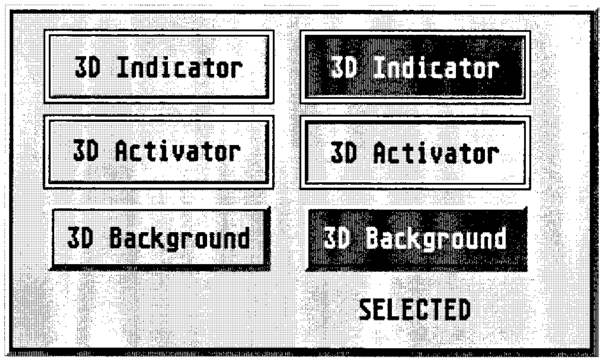

figure #2

If bit 10 only of the *ob_flags* field is set, then the object is a 3D ACTIVATOR. Activator objects don't have a persistent state, but are usually controls of some sort. The ***OK*** and ***Cancel*** buttons in a dialog would normally be activator objects. The new 3D window parts are activator objects.

If both bits 9 and 10 of the *ob_flags* field are set, then the object is a 3D BACKGROUND. These objects are usually not selectable and do not normally display 3D effects, but they are drawn using the system settings for

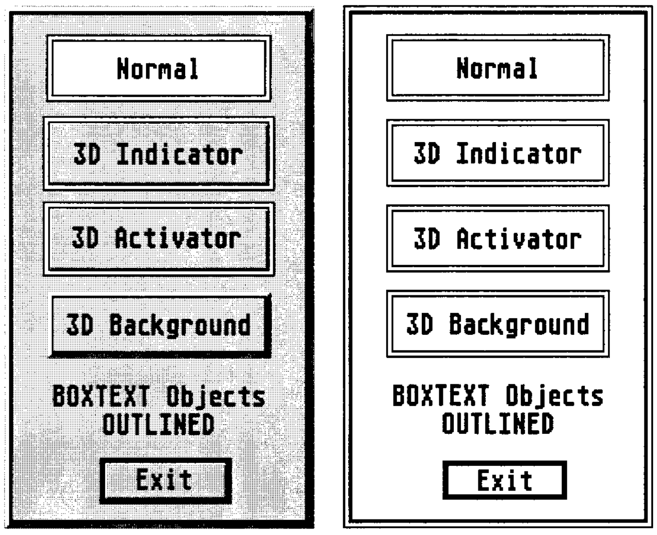

figure #1

Listing #1

```
/* Bit masks to use in accessing the 3D effects bits of */
/* the ob_flags field of an OBJECT */

#define     FL3DMASK    0x0600
#define     FL3DNONE    0x0000
#define     FL3DIND     0x0200
#define     FL3DACT     0x0600
#define     FL3DBAK     0x0400

/* These macros can be used to inquire the 3D status */
/* of an object  They are used like this: */
/* obj_is_3d = Is3dObj(objtree,OBJINDEX); */

#define Is3dObj(a,b) (a[b].ob_state & FL3DMASK)
#define Is3dInd(a,b) (a[b].ob_state & FL3DIND)
#define Is3dAct(a,b) ((a[b].ob_state & FL3DACT) == FL3DACT)
#define Is3DBack(a,b) (a[b].ob_state & FL3DBAK)

/* These macros can be used to set the 3D flags for an */
/* object They are used like this: */
/* Make3dInd( objtree, OBJINDEX ); */

#define Make3dIndr(a,b) (a[b].ob_state |= FL3DIND)
#define Make3dAct(a,b) (a[b].ob_state |= FL3DACT)
#define Make3dBack(a,b) (a[b].ob_state |= FL3DBACK)

/* ob_swhich values for use with objc_sysvar() function */

#define     LK3DIND     1
#define     LK3DACT     2
#define     INDBUTCOL   3
#define     ACTBUTCOL   4
#define     BACKGRCOL   5
#define     AD3DVALUE   6
```

3D object colors. The only 3D effects applied to background objects is that objects with the OUTLINED state set will appear to be raised above the objects underneath them, and that disabled background objects are shaded out using the background 3D color instead of a white dither pattern.

Listing #1 shows some definitions for bit masks and macros that access the *ob_flags* field to inquire if an object has the 3D flags set or to set them.

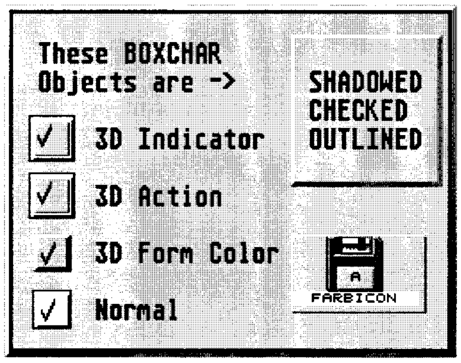

figure #3,
Sample Dialog with 3D Objects With MultiTOS

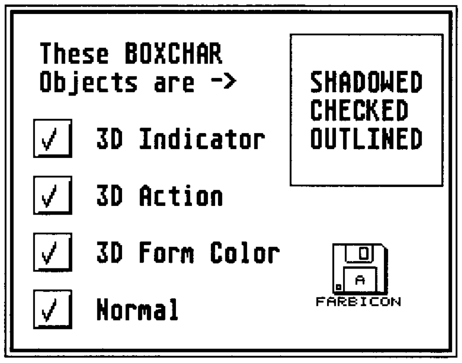

figure #4,
Same dialog as figure #3 with TOS v3.06

You may notice that 3D objects are a little larger than normal objects. One reason for this is that they wouldn't be able to fully contain a text string if they were made to occupy the same exact size as normal objects. The new ***objc_sysvar()*** function can be used to inquire how much extra space is used for 3D objects, as well as other information such as what colors are used to draw 3D objects. You can use this information to adjust object positions as necessary at runtime if you have objects that have to be positioned immediately next to each other, as with a scroll bar and arrow buttons like in the file selector.

The objc_sysvar() Function

```
ob_ret = objc_sysvar( short ob_smode, short ob_swhich,
                      short ob_sivall, short ob_sival2,
                      short *ob_soval1, short *ob_soval2);
```

This new AES function allows an application to set or inquire the colors and effects used for drawing 3D objects. Applications should not change 3D colors or effects except at the request of the user, because all such changes are global and affect all processes.

When the 3D flags are set, some objects become a little wider and a little taller. If your program uses objects which are immediately next to one another, like some arrows and a scroll bar, it can use the ***objc_sysvar()*** function to inquire the amount of extra room used by 3D objects and then use that information to adjust the object positions at runtime.

Listings #2 contains a binding for the ***objc_sysvar()*** function for Lattice C v5.5, Alcyon C, and Mark Williams C. You also need Listing #3 (AESBINDR.H) and Listing #4 (AESCALL.H) from the article ***Popup Menus & Hierarchical Submenus, Part 1,*** which appears earlier in this issue. Mark Williams C and Alcyon C use the same style for bindings, so if you are using Alcyon C, either define the MWC value or change the listing to indicate Alcyon.

If the *ob_smode* parameter is 0, then the call inquires the current attributes. If it's 1, then the call sets new attributes.

The *ob_swhich* parameter indicates which attributes are being accessed. Listing #1 includes some definitions for the meanings of the *ob_swhich* parameter. The *ob_swhich* parameter also affects the meaning of the other values, as follows:

LK3DIND -- Get/set attributes for indicator objects.

If *ob_smode* is 0, then returns current attributes in *ob_soval1* & *ob_soval2*:

```
ob_soval1 = indicator object text moves when object
            is selected (1 = true, 0 = false)

ob_soval2 = Indicator object changes color when
            selected (1 = true, 0 = false)
```

If ob_smode is 1, then sets new attributes from values in ob_sival1 & ob_sival2:

```
ob_sival1 = indicator object text moves when object
            is selected (1 = true, 0 = false)

ob_sival2 = Indicator object changes color when
            selected (1 = true, 0 = false)
```

LK3DACT -- Get/Set attributes for activator objects. The meanings of *ob_soval1*, *ob_soval2*, *ob_sival1*, and *ob_sival2* are the same as for LK3DIND, except that they apply to activator objects rather than indicator objects. The defaults for activator objects is *ob_sival1* = 0 and *ob_sival1* = 1.

INDBUTCOL -- Get/Set the color for indicator objects. This is the color which hollow, white indicator objects (e.g. buttons) will be drawn in instead of white. If *ob_smode* is 0, then *ob_soval1* is the current color index of the default indicator object color. If *ob_smode* is 1, then *ob_sival1* is the new color index for indicator objects.

ACTBUTCOL -- Get/set default color for activator objects. Same as **INDBUTCOL**, but applies to activators rather than indicators.

BACKGRCOL -- Get/set default color for 3D background objects. Same as **INDBUTCOL**, but applies only to 3D background objects.

AD3DVALUE -- Get pixel adjustments for 3D indicators and activators. This is inquire-only, so *ob_smode* must be 0. The *ob_soval1* will indicate the number of pixels by which 3D indicators and activators are expanded on each side horizontally (to accomodate 3D effects), and the *ob_soval2* value is the number of pixels by which they are expanded vertically. Remember that this adjustment is applied to each side of the object, so the objects width or height is increased by twice this amount. Background 3D objects never change in size.

The *ob_sret* return value is zero if an error has occured, usually because an illegal value was given for *ob_swhich* or for *ob_smode.* If the value is non-zero, then the function succeeded.

3D Windows

The first place you are likely to notice the new 3D capabilities is in a window from the GEM Desktop. Window gadgets like the close button, scroll arrows, etc., are actually part of an OBJECT tree. Because of this, they can easily take advantage of the new 3D capabilities. This all happens automatically without any intervention by the user or by a program. However there are a few things worth noting.

First of all, as previously noted, a 3D object takes up more room on screen. This means that the work area of a window with 3D objects will be a few pixels smaller than a window without 3D objects. If your programs are written correctly so that they adjust themselves to whatever screen or window size is available, this will not matter to you.

Secondly, if you have dialog boxes with scroll bars and arrows, or if you have objects like this in the work area of your window, you will probably want to make them use the new 3D ACTIVATOR objects so that their appearance will be consistent with the system's window gadgets. As mentioned earlier, you will probably have to adjust the position and/or size of these objects slightly at runtime in order to account for the slightly larger size of 3D objects.

Listing #2- AESBIND.C

```
#include "aesbindr.h"

#ifdef MWC
short *_AESpb[] = { global, control, int_in, int_out, addr_in, addr_out };
#endif

short
objc_sysvar( ob_smode, ob_swhich, ob_sival1, ob_sival2, ob_soval1, ob_soval2 )
short ob_smode, ob_swhich, ob_sival1, ob_sival2, *ob_soval1, *ob_soval2;
{
        _AEScontrol[0] = 48;
        _AEScontrol[1] = 4;
        _AEScontrol[2] = 3;
        _AEScontrol[3] = 0;
        _AEScontrol[4] = 0;
        _AESintin[0] = ob_smode;
        _AESintin[1] = ob_swhich;
        _AESintin[0] = ob_sival1;
        _AESintin[1] = ob_sival2;

        aes( _AESpb );

        *ob_soval1 = _AESintout[1];
        *ob_soval2 = _AESintout[2];

        return( _AESintout[0] );
}
```

Atari TOS Q & A

Mike Fulton

Q: When my program uses ***Pexec()*** to run a GEM-based program, that program gets the wrong information from the ***shel_read()*** function. What it gets is the information for my program instead and the other program can't find its preference files and so forth. I can't use ***shel_write()*** because it doesn't want to run the program until the shell quits. What do I have to do to make this work?

A: The problem here is that the ***shel_read()*** function always returns the last piece of information used with the ***shel_write()*** function. Since the your shell program is doing just a ***Pexec(),*** the last ***shel_write()*** call is probably the one that was used by the desktop to run your program, so that's what the ***shel_read()*** function returns.

When running under MultiTOS, you should normally use ***shel_write()*** instead of ***Pexec()*** . It will now start another program immediately instead of waiting for the first one to quit. For older TOS versions, Atari Developer Ian Lapore has figured out a good solution to this problem, shown in the code example below.*

```
shel_write( 1, 1, 1, "NEWPROG.PRG", commandline );
Pexec( 0, "NEWPROG.PRG", commandline, environment );
shel_write( 0, 1, 1, "NEWPROG.PRG", commandline );
```

The value of 1 in the first parameter for the first ***shel_write()*** means "run another application when this application exits". This call gives AES the proper information so that when "NEWPROG.PRG" does its ***shel_read()*** call, it gets the right results.

Next comes a Pexec() call that actually executes "NEWPROG.PRG" and waits for it to quit. When "NEWPROG.PRG" quits and your program gets control back, we have a problem. When the user quits your program, the AES is going to want to run "NEWPROG.PRG" again, because we did a ***shel_write()*** call asking it to do so.

To get around this, after the ***Pexec()*** call, you do another ***shel_write()*** call. This time we use a value of 0 for the first parameter, which means "exit and return to the desktop when the current application quits". This cancels out the previous ***shel_write()*** call.

**** This method should only be used when not running under MultiTOS.***

Atari Developer News

ATARI.RSC STAFF

Lattice C v5.5 Now Available

Lattice C v5.5, an ANSI C compiler with a GEM-based integrated development environment, is now available directly from Atari Corp. to commercial developers in North America. The price is $199.95 (U.S.) plus sales tax and shipping. Contact Atari Developer Support to order.

SpeedoGDOS Now Available

SpeedoGDOS and 14 bundledfonts are now available for licensing to commercial developers. It must be sold strictly in conjunction with your GDOS-related software, either bundled or as a customer-service provided "enhancement". If you are interested, please send EMAIL to Bill Rehbock (GEnie: B.REHBOCK, Compuserve: 75300,1606) or fax (408) 745-2088.

MultiTOS v1.01 Now Available

MultiTOS release v1.01 is now available. Commercial-Level developers may download MultiTOS from Library 6 of the ATARI.RSC Developer Roundtable on GEnie. (Library 6 is restricted to commercial developers only, send EMAIL online to MIKE-FULTON or ATARIDEV to request admission.) Associate-Level developers may purchase MultiTOS for $59.95 through Atari Developer Support.

MultiTOS documentation is available to all developers in Library 10 of the ATARI.RSC Developer Roundtable on GEnie or in Library 7 of the ATARIPRO forum on Compuserve.

Drag & Drop Protocol

One of the new features of MultiTOS is a protocol for drag and drop operations. This is where you pick up an object of some kind from within one application, drag it across the desktop, and drop it onto a window or icon belonging to another application. The object can be something like a file icon, a graphic object, and so on.

When the user drags and releases an object, the program that owns the object determines the window ID and window owner for the spot where the object was dropped. If the owner is another application, then the application sends an AP_DRAGDROP message to that application.

If the receiving application is aware of the drag and drop protocol, then it and the originating application send messages back and forth to figure out what sort of data types are acceptable and so on.

The full specification for the drag and drop protocol is given in the DRAGDROP.DOC file which is part of the MultiTOS documentation. Drag and Drop will also be discussed more fully in an upcoming issue of ATARI.RSC.

Minimizing Windows and Applications

Another new protocol outlines how to minimize and maximize windows and/or applications. That means the current window or application is replaced by a small window the size of a desktop icon which is then placed on the desktop.

The idea is to give the user the capability to hide an application or window temporarily so that other applications or desk accessories may be more easily accessed. A minimized window can later be maximized back to the previous when the user needs to access it again.

An outline of the steps for minimizing and maximizing your windows and/or applications is available in the MINIMAXI.DOC file. This file is part of the MultiTOS documentation available online. Please note, however, that this technique is not MultiTOS-specific and can also be used on older versions of TOS.

A future issue of ATARI.RSC will contain more information on this subject.

New GEM AES Features

There are a number of new GEM AES features in MultiTOS that we haven't mentioned yet. Below is a short list of some of the new features. These features and others are documented more fully in the GEM.DOC file that is included in the MultiTOS developer distribution.

appl_search() - You now have the capability to search for other programs which are running at the moment.

appl_getinfo() - Gives you information about the AES such as what fonts it is using, the number of colors supported by OBJECTs, the current language preference, etc..

graf_mouse() - Now allows you to save and restore the mouse shape.

shel_write() - Now allows you to execute a program immediately, without waiting for the current program to quit. Now allows you to execute your choice of application, accessory, or TOS/TTP program, or it can figure it out for you from the program's extension. You can even now specify the program's GEMDOS Environment.

Unless you need one of the more exotic modes of ***Pexec()***, you should now use ***shel_write()*** instead.

wind_get() - There are now several new types of information available via the ***wind_get()*** function,

(continued on page 12)

Atari Developer News
(continued from page 11)

including the Application ID of a window's owner.

wind_set() - There are some new window features which can be accessed via the ***wind_set()*** function. For example, you can tell the AES that clicking the mouse in a window will generate a button event when the window is not on top, instead of a generating a **WM_TOPPED** message. This is useful for creating special toolbox windows that go along with your document windows.

A couple of tips regarding toolbox windows: If you get a topped message for a document window, then first top the toolbox window, then the document window specified in the message. (You can still manually top a toolbox window even if it won't generate **WM_TOPPED** messages from mouse clicks.) It's also a good idea to either make the toolbox window go away if you have no document windows open or at least reset it so that mouse clicks generate **WM_TOPPED** messages again.

It's also now possible to send a window to the bottom of all open windows.

wind_update() - You can now test if another program already has control of the system. If you are using ***wind_update()*** in order to update a status display for something like downloading a file, sending a fax, compiling source code, and so forth, you can now determine if somebody already has control of the system. If some other program does have control already, you can skip the update to the status display. This allows your program to continue with its main task instead of having it wait until the other program releases control. The next issue of ATARI.RSC will contain a tutorial on using the ***wind_update()*** call correctly, including this new feature.

New AES messages include:

WM_UNTOPPED - Window is being untopped

WM_ONTOP - Window has come to the top through no action of its own, such as when the previous top window is closed.

AP_TERM - Please shut down your program and quit. This would be sent in circumstances such as (but not limited to) when the user selects a new video mode from the desktop.

SH_WDRAW - This message is sent from applications to the desktop to inform it that it should update windows for a particular drive. (Because the application has created or deleted files.)

Window Gadgets - All window gadgets such as the move bar, scroll arrows & slider area, sizer, etc., are now fully active even when a window is not topped. That means it's now possible to get messages such as **WM_ARROWED**, **WM_MOVED**, **WM_SIZED**, etc., even when the window is not on top.

Some applications always assume the window must be on top when such a message is received. Also, some programs that allow multiple windows always apply such messages to its topmost window, even if that's not the window specified in the message. If your application does either of these things, then it will need to be updated. (Please note that the AES manual has never specified that these messages will only happen to the top window. That's why the window handle is part of the message.)

Dealer Sign-Up

If you don't have a dealer in your area that carries Atari computers, but you know of a dealer that you would like to see selling the Falcon030, we'd like to know about it so we can arrange a visit by one of our dealer sales representatives. Please mail or fax the dealer's name and address to us at the address or fax number found on page 2. Please address it to the attention of Bob Brodie and James Grunke.

Atari Falcon030 Now Shipping to Dealers

As this issue of ATARI.RSC goes to press, the new Atari Falcon030 computer has started shipping to dealers in Europe and North America.

Owners of new Falcon030s will be anxious to try out some software that takes advantage of the machine, so if you've been holding back the release of your product waiting for the machine to be available, now's the time to release the program and get the word out.

Atari Computer Corporation
1196 Borregas Ave.
Sunnyvale, CA 94089-1302

ATARI.RSC
The Developer's Resource