Chapter 20
EZMenu System

Tutorial

The Amiga provides a very sophisticated pull down menu system that greatly enhances the usability of an application. A group of menus can be linked together and associated with a specific window. Each menu has a list of items that the user can select. Each item can be either text or a bitmapped image. Menu picks are obtained by requesting IDCMP messages. When an application receives a menu pick, it executes the appropriate piece of code. A detailed explanation of how this system works can be found in the Intuition Manual.

Using JForth, you can access all of the features of Intuition Menus. Because the Amiga has so many options, however, this can be a somewhat difficult process. We decided to offer a simplified Menu interface that will suffice for most applications. We assume that you would like one or more menus with simple text items. The EZMENU system will build Menu, MenuItem, and IntuiText structures based on this assumption. If you want more fancy menus, you can still use the EZMENU system, but you will need to tweak things a bit.

An example of using the EZMenu system can be found in the file JD:DEMO_MENUS.

EZMenu Structure

The EZMenu system is based on the use of a special JForth structure called an "EZMenu". The EZMenu contains an Intuition Menu structure, a pointer to an array of IntuiText items, a pointer to an array of CFAs (function addresses), and a count of how many items are in the EZMenu. It is defined in the file JU:AMIGA_MENUS . To use the EZMenu system, you first create a number of EZMenu structures. The address of these structures is then passed to the EZMenu routines. By combining the EZMenu routines with the low level Menu routines described later, you can have total control over the menus.

AMENU Program

Let's write a simple menu program that opens a window and attaches to it one menu with three items. We will then scan for Amiga events and process any MENUPICK events. Once you are comfortable with the essentials covered by this program you can explore JD:DEMO_MENUS and JA:EZWALKER.F for more extensive examples.

First we will need to compile the graphics and event support which we studied earlier and the EZMenu code. I recommend typing this program into a file as we go since it is a little long for keyboard entry.

```
INCLUDE? NEWWINDOW.SETUP JU:AMIGA_GRAPH
INCLUDE? EV.GETCLASS     JU:AMIGA_EVENTS
INCLUDE? EZMENU JU:AMIGA_MENUS
```

In the file, after the INCLUDE?s, let's put a call to ANEW. This will help us when we reload this program by automatically forgetting the previous version. We will then create our own EZMenu structure to use with the EZMenu routines. You will need one EZMenu structure for each menu.

```
ANEW TASK-AMENU
EZMENU MY-MENU
```

Now let's write a word that will initialize our menu. We need to allocate room for 3 menu items. EZMENU.ALLOC? will allocate all the needed MenuItem and IntuiText structures needed for the menu. EZMENU.SETUP will link all of these structures together and give the menu a name. The 0

indicates that this is the 0th menu for the menu. Menus and menu items are all numbered starting with zero. We then use EZMENU.TEXT! to give each menu item a name Finally we assign a Command Sequence Key to the "Quit" item. This will allow us to quit by simply holding down the Right Amiga Key and hitting 'Q'.

```
: MY-MENU.INIT  ( -- error? , INITIALIZE MENU )
\ Allocate space for 3 menu items
   3 MY-MENU EZMENU.ALLOC?  \ returns 0 if it fails
   IF
\ Set name of menu and position in list.
      0" Project" 0 MY-MENU EZMENU.SETUP
\ Define the text for each menu item.
      0" Open" 0 MY-MENU EZMENU.TEXT!
      0" Close" 1 MY-MENU EZMENU.TEXT!
      0" Quit"  2 MY-MENU EZMENU.TEXT!
      ASCII Q 2 MY-MENU EZMENU.COMMSEQ!
      FALSE \ flag for NO error
   ELSE
      ." MY-MENU.INIT - Insufficient Memory!" CR
      TRUE  \ error flag
   THEN
;
```

This is all that is required to define the appearance of a menu. If you want to get fancy you could modify some of the menu items to add checkmarks, graphic images, subitems, etc. See EZMENU.ITEM[] for details on how to access individual items.

We now need a word that will process a Menu Pick when it occurs. The Amiga will generate a "menucode" that indicates which item was picked. By using ITEMNUM() , MENUNUM() , and SUBNUM() you can determine exactly which item or subitem in any menu was picked. We only have one menu and no subitems so we only need ITEMNUM().

Let's keep a flag variable that will tell us when to stop. If we hit "Quit" from the menu we can just turn that variable on. If we hit the other two items, just output a message. Nothing fancy. That's for you to add!

```
VARIABLE QUIT-NOW    ( time to stop? )
: DO.WHATEVER ( menucode -- , act on menu item chosen )
    DUP MENUNULL =
    IF DROP   ( not a complete menu pick )
    ELSE ( -- menucode )
        ITEMNUM()  ( -- item# )
        CASE
            0 OF ." Open File!" CR ENDOF
            1 OF ." Close File!" CR ENDOF
            2 OF ." Quit!" CR QUIT-NOW ON ENDOF
        ENDCASE
    THEN
;
```

The program will probably be receiving more than just Menu Picks. There will be Mouse Clicks, CloseWIndow events, etc. We need to treat the differently. This next word passes Menu Picks to DO.WHATEVER and also turns on the quit flag if the CloseWindow Gadget is hit. The information on what menu item was picked is stored in EV-LAST-CODE by the EV.GETCLASS word. This word is described in the section on Event Handling.

```
\ Process IDCMP events.
: HANDLE.EVENT ( eventclass -- )
    CASE
\ Perform Menu actions
        MENUPICK
        OF  EV-LAST-CODE @ DO.WHATEVER
        ENDOF
\ Set quit flag if CLOSEBOX hit.
        CLOSEWINDOW
        OF QUIT-NOW ON
        ENDOF
    ENDCASE
;
```

This last word only handles one event. We, therefore, need a loop that will scan for events in our application window and pass them to HANDLE.EVENT for processing. We use GR-CURWINDOW, which holds a pointer to the current window, to get events from. The loop will quit when the QUIT-NOW flag is turned on.

```
: LOOP.MENU ( -- , poll for events )
   QUIT-NOW OFF
   BEGIN
\ wait for an event so we don't tie up the Amiga
      GR-CURWINDOW @ EV.WAIT
\ Check for events in the current window.
      GR-CURWINDOW @ EV.GETCLASS ?DUP
      IF HANDLE.EVENT
      THEN
      QUIT-NOW @
   UNTIL
;
```

Now we just need to tie all this together. We start by initializing graphics and opening a window. I then initialize the menu and attach it to our window using SetMenuStrip(). When I terminate I detach the menu with ClearMenuStrip() and also free the memory declared by EZMENU.ALLOC. I then run the whole thing from the AMENU word. I have found that this style of organizing a program, with a clear INIT and TERM helps reduce bugs and makes for more readable code. Note how the error codes from GR.OPENCURW and MY-MENU.INIT propagate up to the highest level.

```
   NEWWINDOW MYNW
: AMENU.INIT ( -- error? , set everything up )
   TRUE \ set default error return
   GR.INIT
   MYNW NEWWINDOW.SETUP
   MYNW GR.OPENCURW  \ returns &window if successful
   IF
```

```
\ Initialize menu and attach to window.
      MY-MENU.INIT 0=
      IF
         GR-CURWINDOW @ MY-MENU SETMENUSTRIP()
         DROP FALSE  \ replace TRUE since no error
      THEN
   THEN
;
: AMENU.TERM ( -- , clean up menus and close window. )
   GR-CURWINDOW @ ?DUP
   IF
      CLEARMENUSTRIP()
   THEN
   MY-MENU EZMENU.FREE
   GR.CLOSECURW
   GR.TERM
;
: AMENU ( -- , do it all)
   AMENU.INIT 0=  \ Everything OK?
   IF
      LOOP.MENU
   THEN
   AMENU.TERM
;
cr ." Enter:    AMENU    to see demo." cr
```

I like to close my program files with a reminder of what to enter to run the program.

EZMenu Glossary

For the following words, the parameter 'ezmenu' refers to the address of an EZMenu structure. All of the addresses passed to and received from these routines are JForth relative addresses unless otherwise specified. All numbering is zero based. The first MenuItem, therefore, has an item# of 0. For the following examples, assume that you have created two menus, as follows:

```
EZMENU MAINMENU
EZMENU EDITMENU
```

EZMENU (<name> --input-- , Create an EZMenu structure.)

EZMENU.ALLOC? (#items ezmenu -- &items | 0)

Allocate memory needed for the MenuItems, the Intuitext items, and the CFA array. The first MenuItem is then linked to the Menu. If any of the allocations fail, a zero is returned. Otherwise the address of the items is returned. You should check the return value before proceeding with any other EZMENU calls.

```
\ Allocate space for 6 menu items.
6 MAINMENU EZMENU.ALLOC?  .
```

EZMENU.CFA[] (item# &ezmenu -- &cfa, Address of cfa)

The EZMenu system maintains an array of CFAs associated with each menu. A CFA is Forths equivalent to 'C's pointer to a function. The array is initially filled with the CFA of NOOP. You can use EZMENU.EXEC to execute the appropriate word after a menupick. The CFA of a word can

be obtained by "ticking" a word with the word ' . Any word's CFA can be placed in this array as long as it doesn't take from or leave anything on the stack.

```
\ Set the CFA for menu item 3
: ACT3 ( -- , No parameters allowed.)
    ." Action 3" CR
;
 ' ACT3  3 EZMENU.CFA[]  ( get address )  ! ( store )
```

EZMENU.COMMSEQ! (char item# &ezmenu -- , Set command key.)

You can specify that a menu item be 'picked' by hitting the right Amiga key and a special character together. This provides a handy shortcut to a menu action.

```
\ Set Command Sequence key to 'W'.
ASCII W  3 MAINMENU EZMENU.COMMSEQ!
```

EZMENU.EXCLUDE!   (mask item# &ezmenu -- , set auto-exclusion)

Selecting one MenuItem can automatically cause others to become unchecked. See the Intuition Manual, page 6-6, for details. Note that this word also sets the CHECKIT flag.

EZMENU.EXEC   (menucode menustrip -- , run menuitem's action)

The MENUCODE is obtained from Intuition. It is placed in the JForth variable EV-LAST-CODE by a call to EV.GETCLASS . The menustrip is the same as the address of the first EZMenu in the list.

EZMENU.FREE (&ezmenu -- , Free memory from EZMENU.ALLOC)

EZMENU.ITEM[] (item# &ezmenu -- &item , item address)

If you want to modify a MenuItem structure for special handling, use this word to obtain it's address.

EZMENU.TEXT[] (item# &ezmenu -- &intuitext , text address)

If you want to modify a MenuItem's associated IntuiText structure, use this word to obtain it's address.

EZMENU.TEXT! (text0 item# &ezmenu -- , Set text for MenuItem)

The text must be a NUL terminated string such as that created by 0" . (That character in front of the quote is a zero. In some fonts this is not obvious.)

```
\ Set menu item text.
0" Write"  3  MAINMENU  EZMENU.TEXT!
```

EZMENU.SET.FLAG (flag item# &ezmenu -- , OR flag with existing)

You can set your own bits in the flags member of a MenuItem structure with this word.

```
\ Put checkmark beside item# 2 .
CHECKED 2 MAINMENU  EZMENU.SET.FLAG
```

EZMENU.SETUP (name0 menu# &ezmenu -- ,Set default values)

This word initializes the values in the Menu, MenuItem, and IntuiText structures associated with an EZMenu. It then links together all the pieces for Intuition to use. It must be executed after EZMENU.ALLOC and before any calls to EZMENU.TEXT! , EZMENU.SET.FLAG, EZMENU.COMMSEQ! , etc. The first parameter is a NUL terminated string that is the name for the menu. The second parameter is the menu's intended position in the MenuStrip. Remember the first menu is number 0. You may want to change the default settings before calling this routine. See below.

EZMENU.SUBMENU! (&submenu item# &ezmenu --)

Set submenu for this item. The example in JD:DEMO_MENUS uses submenus.

EZSUBMENU.SETUP (&ezmenu -- , set defaults and links)

EZMENU.SETITEM (0name cfa char|0 item# &ezmenu --)

Sets the name, cfa and command character for a menu item. Provided for convenience.

EZMenu Default Settings

The EZMenu system uses default settings for many of the Menu and Menuitem parameters. These are kept in variables that you can change before calling the EZMenu routines.

INTUITEXT-DEFLEFT (--- addr ,default left edge of IntuiText item)

MENU-DEFLEFT (--- addr , default left edge of a menu)

MENU-DEFWIDTH (--- addr , default width of a menu)

MENUITEM-DEFLEFT (--- addr , default left edge of a menu item)

MENUITEM-DEFWIDTH (-- addr , default width of a menu item)

Low Level Menu Support

The EZMenu system uses some of these words to perform it's functions. Several of these words will need to be called by your application directly.

The following three words are used by EZMENU.SETUP to initialize the Amiga structures needed to use menus.

INTUITEXT.SETUP (&intuitext -- , Set defaults for IntuiText)

ITEMNUM() (menucode -- item# , parse code from event handler)

MENU.LINKTO (menu1 menu2 -- , Make menu2 follow menu1)

A menustrip can be built by linking a number of menus together into a linked list.

MENU.MIS># (subitem# item# menu# -- menunum , Calc MENUNUM)

Many Amiga routines use this compounded menu number.

MENU.NTH (N &menustrip -- &Nth_Menu , Traverse Menustrip)

Follows links in menustrip's linked list to find Nth menu.

MENU.SETUP (name0 menu# &menu -- , Set defaults for Menu)

See EZMENU.SETUP for description of parameters.

MENUITEM.SETUP (item# &menuitem -- , Set defaults for MenuItem)

MENUNUM() (menucode -- menu# , parse code from event handler)

SUBNUM() (menucode -- subitem# , parse code from event handler)

The following words make calls to the Intuition library.

SetMenuStrip() (&window &menustrip -- , Attach menus to window)

The menustrip is the address of the first menu in the linked list.

ClearMenuStrip() (&window -- , Remove the menus from a window.)

OnMenu() (window menunum -- , Enable part of menu)

These routines are used when a menu is currently attached to a window. The menunum can be created using MENU.MIS># .

OffMenu() (window menunum -- , Disable part of menu)