

Diamond GOS Developer's Kit

First printing

(C) Copyright 1990
REEVE Software

REEVE Software
29W150 Old Farm Lane
Warrenville, IL 60555
(708) 393-2317

Diamond Develop

Introduction

Diamond GOS is Reeve Software's powerful Graphical Operating System for Atari 800/XL/XE computers. This manual documents the Diamond environment from a programming perspective and provides detailed information on how to use Diamond's graphical abilities in your own programs. The manual has been organized to provide detailed descriptions of each of Diamond's routines in groups so that all routines dealing with menus are together...etc. At the end of the manual each of Diamond's routines is summarized for quick reference.

Note: The disk included with Diamond Develop has been written in DOS 2.X format. If you are using either SpartaDOS or DOS XE then all of the files from this disk must be copied to a disk formatted by the DOS of your choice.

Index

Memory Layout	3
Functions	3
Data Structures	12
Mouse Drivers	15
Memory Drivers	15
Desk Accessories	15
Other Included Files	17
Appendix I: Function Reference	18
Appendix II: Memory Locations	28
Appendix III: Languages	31
Conclusion	32

Memory Layout

One of the biggest problems with a Graphical Operating System is the amount of memory that is consumed by all the fancy graphics used. Normally programs that deal with data use a text mode to display numbers and figured, but Diamond must, at ALL times, use a graphics screen that uses roughly 8K of memory, while a text only screen consumes less than 1K.

Diamond resides in a 64K cartridge. One of the most amazing things is that the amount of data in the Diamond GOS cartridge is EQUAL to the amount of memory that can be addressed by the 6502 chip in your Atari 800/XL/XE computer. Of the 64K that your Diamond cartridge holds there are 8 8K banks of memory that are swapped in and out of memory at locations \$A000-\$BFFF. The cartridge itself can also be banked out allowing us to use the 8K of memory under the cartridge's memory so we don't really lose this memory, but are merely inconvenienced in using it.

Diamond also uses memory from \$8C00-\$9FFF for miscellaneous storage locations, system variables, and for calling Diamond. When Diamond is being used, screen memory resides at locations \$6000-\$7FFF (Actually it starts at \$6035). This leaves us with 1 big chunk of memory to program with which starts at LoMem and goes through to screen memory. Location \$8000-\$8BFF also gives us an area to play with (we have developed a Diamond File Selector which is normally placed here and is included in this package). Locations \$A000-\$BFFF can also be used with the proper technique which gives us roughly 27K to program with.

Diamond uses memory from locations \$9A80 to \$9AD2 as system variables. These values are used in conjunction with Diamond's various operations as listed in Appendix II of this document.

Calling Diamond

To call one of Diamond's routines using assembly or machine language ~~we~~ simply load the accumulator with the routine number and call location \$8E00. To pass parameters to these routines Diamond uses 16 pseudo registers that will be referred to as B0-B7 and W0-W7 (they actually reside at memory locations \$80-\$97 in page 0). The B registers are byte sized and the W registers are words (two bytes). A call to Diamond in assembly language might look like this:

```
LDA #0 ;Select Low Res
STA B0
LDA #0 ;INIT routine
JSR $8E00 ;Call Diamond
```

obviously the second LDA is not needed here as the accumulator already contains 0, but this will be the general format for Diamond calls.

Diamond Functions

Diamond GOS consists of several routines to manipulate and poll the drop-down menus, windows, icons...etc. We have broken these routines down into ten groups and they are listed with their name followed by a routine number (in parenthesis) and an explanation of when, why, and how you would use this routine.

System Functions

The following are considered to be Diamond's System Functions as they are general purpose routines that didn't seem to fit into any other group. The first three are pertinent to initializing and terminating a Diamond GOS application.

Diamond Develop

INIT (0) is called IMMEDIATELY before doing anything else to initialize Diamond for your application. To call INIT simply place the resolution that you wish to use in B0 (0=High (the norm), 1=Low) and call Diamond. Sample assembler code to do this would appear as:

```
LDA #0 ;High resolution
STA B0
JSR DIAMOND ;Function=0 (0 already in Accum)
```

EXIT (1) is called to terminate your Diamond application in the same manner as INIT is used to initialize Diamond. EXIT receives no parameters.

EXECDESKTOP (48) Once EXIT has been called an application will usually return to the DeskTop via this routine. This routine also receives no parameters. The following code fragment will terminate your Diamond GOS application and return to the DeskTop:

```
LDA #1 ;Exit function
JSR DIAMOND
LDA #48 ;Exec DeskTop
JSR DIAMOND
```

In summary a Diamond GOS application should call INIT first. The actual program will then follow. After the program is done call EXIT and then EXECDESKTOP to make a legal exit to the Diamond DeskTop.

BINLOAD (53) is used to load other programs into memory. It loads standard DOS 2.X format files (as do all (most?) other DOS's). If no Run address is supplied (at \$2E0) then once BINLOAD loads the selected file it will return control to the calling program. BINLOAD can therefore be used to load and run Diamond applications (as it is used by the Diamond DeskTop) or to load code overlays as is used by Diamond Paint for printing.

A code overlay is used when a program is too big to fit in memory at once...BINLOAD can be called to load a block of code, that block can be executed, BINLOAD loads another routine over the previously loaded routine, executes it...etc. To call BINLOAD B0 should be set to whether a Diamond application is being loaded (0) or told to disable Diamond (1), and W0 points to a filename.

```
LDA #0 ;Diamond Enabled
STA B0
LDA #< FILENAME ;Get Addr of file
STA W0
LDA #> FILENAME
STA W0+1
LDA #53 ;BINLOAD Function #
JSR DIAMOND
...
FILENAME .BYTE "D1:MYPROG.OBJ", $9B ;File to load
```

TOGGLEZERO (3) is, quite frankly, useless. It is used by our windowing routines and simply swaps all registers (B0-B7/W0-W7) with a built in buffer (hence two successive calls to TOGGLEZERO accomplishes nothing as the original swap will be swapped back with itself).

The following four functions relate to Diamond GOS's cursor which is simply a vertical bar (or block) on the screen. This cursor is usually used to indicate where data entry is to take place from the keyboard.

CURSORON (34) turns Diamond GOS's cursor on. No parameters are needed.

Diamond Develop

CURSOROFF (35) turns Diamond GOS's cursor off. No parameters are needed.

DEFCURSOR (36) sets up the type of cursor Diamond GOS is to use. Register B0 will contain a bit-pattern that defines the shape of the cursor, B1 the height (in pixels) of the cursor, and B2 the flash rate of the cursor in 60ths of a second.

```
LDA #$80 ;Bit pattern = 10000000 (vertical bar)
STA B0
LDA #8 ;8 bit patterns high
STA B1
LDA #15 ;Flash every 1/4th of a second.
STA B2
LDA #36 ;DEFCURSOR routine
JSR DIAMOND
```

MOVECURSOR (37) moves the cursor to a given location on the screen. B0 is to contain the new X position of the cursor, and B1 its new Y position.

The following three routines deal with calculations for Diamond. The 6502 does not have its own multiply or divide instructions so we have provided two routines that accomplish these tasks. In addition to this, sometimes it will be necessary to determine the address of a particular location on the screen. This could be accomplished with a multiply by 40 (the screen width) by the Y coordinate and adding the X coordinate to that, but we have supplied a routine called MUL40 to do this for you.

MULTIPLY (39) will multiply the number in W5 by the number in W6 and store the result at W7. Please note that no error reporting for overflow is accounted for and that a bug in this routine results in a system crash when an attempt to multiply by zero occurs.

```
LDA #5 ;Place 5 at W5 and W6
STA W5
STA W6
LDA #0
STA W5+1
STA W6+1
LDA #39 ;Call multiply and 25 will be returned
JSR DIAMOND ;in W7
```

DIVIDE (40) will divide the number in W4 by the number in W5 storing the result in W6 and the remainder in W7. As with multiply no errors are accounted for.

MUL40 (54) receives the X position in B6 (0-39) and the Y position on the screen in B7 and returns the screen address at that location in W7.

The following two routines just didn't seem to fit with anything else so here they are.

(Add Params used for the following routines)
DRAGBOX (45) is used to create a box and allow the mouse to drag that box around the screen until the mouse button is released where it will return the final position of the box. This function is used by Diamond DeskTop when the drive icons and trash can are dragged to a new location.

ADDQUE (38) adds a routine to our own VBI queue. Four slots are available and Diamond GOS uses two of these slots. Any routines added here are cleared whenever Diamond is INITed.

Memory Functions

Memory routines deal with the rapid movement of large blocks of memory. Being as Diamond supports a four byte address range for memory locations so do these routines, and thus moving

Diamond Develop

to and from extra memory no longer requires determining how to write to such memory. These routines are divided into two groups. The first group involves moving between a linear block of memory and screen memory, and the second group involves moving between two linear blocks of memory. (Note: All memory routines return an error code in B7 where \$FF means an error occurred)

For screen memory moves the general register set up will be:

W0,W1 - Pointer to a linear block of memory
B0 - X position on the screen (0-39)
B1 - Y position on the screen (0-191)
B2 - Width of area on screen (in bytes)
B3 - Height of area on screen
B4 - Clipping value for X (cut off point)
B5 - Clipping value for Y
B6 - X Offset
B7 - Y Offset

usually registers B4-B7 will be set to 0 as these options were primarily intended for Diamond's routines.

MOVETOSCREEN (27) moves an area of linear memory onto the screen.

MOVEFROMSCREEN (28) moves an area from the screen into linear memory.

CLEARSCREEN (29) erases an area on the screen. No linear memory address is needed here so fill W0 and W1 with dummy values of \$2020 and \$2020.

INVERTSCREEN (30) inverts an area of the screen. This routine does not need a linear address either so fill W0 and W1 with dummy values.

SWAPSCREEN (41) swaps an area on the screen with an area of linear memory.

FILLSCREEN (55) is similar to clear screen but instead of zeroing the background you can select the 'fill' byte by placing it in W2 (the low byte of W2). Dummy values need to be stored in W0 and W1. A fill byte of 0 would make FILLSCREEN identical to CLEARSCREEN.

For linear memory moves the general layout for registers is:

W0,W1 - Address of the source block
W2,W3 - Address of the destination block
W4 - Size (in bytes) of the move

MOVE (31) moves a block of size (W4) bytes from the source location to the destination location.

ZERO (32) clears (zeros) the destination block. The source block is not used and thus must be filled with the dummy values of \$2020 and \$2020.

SWAP (33) swaps the source and destination blocks.

Note: In order to use these routines effectively please note the three system variables TOTALRAM, SYSPTR, and EXTRA as these variables will reveal how much memory is actually available for Diamond's use.

Diamond Develop

Text Functions

Text routines are used for printing text to the screen. Two routines are supplied for this purpose. Of the two PARAPRNT is more flexible and slower, and SYSDRAW is less flexible and faster.

PARAPRNT (2) is used to print a string of text to the screen. B0 and B1 contain the X and Y offset of the text (usually 0), B2 contains the line spacing between lines (usually 8, as characters are usually 8 pixels tall), W1 the X position of the text (0-319), W2 the Y position of the text (0-191), and W5 the address of the text string to print. The text string must be null (0) terminated and can have several different codes imbedded:

10 - Linefeed

13 - Carriage return

251,byte - Print the following character regardless of value (Use to print control characters)

252,byte - Set style to byte (add the following values: Bold=1, Italic=2, Outline=4, Underline=8, Inverse=16, Light=32, Mirror=64, Reverse=128)

253,word - Set font to address in word (Default font=System)

254,byte - Change height magnification (Default=1)

255,byte - Change width magnification (Default=1)

```
LDA #0 ; Zero offsets
STA B0
STA B1
LDA #8
STA B2 ;Line spacing
LDA #10 ;Print at 10,10
STA W1
STA W2
LDA #0
STA W1+1
STA W2+1
LDA #<STRING ;Point to the string
STA W5
LDA #>STRING
STA W5+1
LDA #2 ;PARAPRNT
JSR DIAMOND
...
STRING .BYTE 252,5 ;Bold-Outline text style
        .BYTE 254,2 ;Double height
        .BYTE "Hello, Everybody!"
        .BYTE 13 ;Return
        .BYTE "Hello, World!"
        .BYTE 0
```

The above example prints the message:

```
Hello, Everybody!
Hello, World!
```

at location 10,10 on the screen using Bold-Outline text with double height magnification.

SYSDRAW (44) allows printing of text, but limits this text to having to start in a column from 0-39 (instead of any pixel location on the screen) and does not allow the text to change styles or be magnified...etc. When calling SYSDRAW W0 is used to point to the string to print, B1 and B2 to

Diamond Develop

the X and Y position of the text, and B3 is used to set the style of the text. Please note that unlike strings used with PARAPRINT that these strings must be terminated with a value of 255.

Drawing Functions

PLOTPOINT (24) plots a point on the screen. B0 contains the X position (0-159), B1 contains the Y position (0-191), B2 the color (0-3), and B3 the mode (0=Normal,1=XOR).

PLOTLINE (25) draws a line on the screen. B0 and B1 contains the X1 and Y1 position, B2 and B3 contain the X2 and Y2 position, B4 the color and B5 the mode. The following example draws a line from 10,10 to 10,30 on the screen.

```
LDA #10 ;From 10,10
STA B0
STA B1
STA B2 ;To 10,30
LDA #30
STA B3
LDA #2 ;Color 2
STA B4
LDA #0 ;Normal mode
STA B5
LDA #25 ;Line function
JSR DIAMOND
```

PLOTBOX (26) draws a box on the screen using the same information passed to PLOTLINE.

Mouse Functions

MOUSEON (11) enables the mouse cursor. This routine receives no parameters.

MOUSEOFF (12) disables the mouse cursor. This routine receives no parameters.

DEFMOUSE (13) defines a new mouse cursor with W0 pointing to the new cursor structure. The following example (which sets the mouse cursor to look like an hourglass) should help clarify this:

```
LDA #<NEWCURSOR
STA W0
LDA #>NEWCURSOR
STA W0+1
LDA #DEFMOUSE
JSR DIAMOND
...
NEWCURSOR
.BYTE 3,3 ;MOUSE HOT SPOT
.BYTE $FF ;11111111
.BYTE $42 ;01000010
.BYTE $3C ;00111100
.BYTE $18 ;00011000
.BYTE $18 ;00011000
.BYTE $24 ;00100100
.BYTE $7E ;01111110
.BYTE $FF ;11111111
```

Icons

The following routines pertain to icons. Icons are pictures that appear on the screen that are used to signify something (i.e. the disk drives and trash can on Diamond DeskTop are icons). Diamond supports a maximum of 32 icons at once.

Diamond Develop

INSTALLICON (4) places an icon on the screen. Once INSTALLICON has been used EVENT will detect events for this icon. When called B0 should indicate which icon is to be used (0-31), B1 should hold the X position (0-39) on the screen, B2 should hold the Y position, B3 the width of the icon (in bytes), B4 the height of the icon, and W0 should point to that icon's image in linear memory. The following would install an icon.

```

LDA #0 ;Use First icon
STA B0
LDA #18 ;Install in the middle of the screen
STA B1
LDA #100
STA B2
LDA #5 ;Width=5
STA B3
LDA #20 ;Height=20
STA B4
LDA #<IMAGE ;Point to image address
STA W0
LDA #>IMAGE
STA W0+1
LDA #4 ;Install
JSR DIAMOND
...
IMAGE .BYTE (5 * 20 = 100 bytes of image data)
```

MOVEICON (5) moves an existing icon to a new position on the screen. When called, B0 should hold the number of the icon to move, B1 the new x position of the icon, and B2 the new Y position of the icon.

REMOVEICON (6) removes an icon and stops any further event from being detected for the icon. When called, B0 should hold the number of the icon to remove.

SHAPEICON (7) allows an icon's shape to be altered for animation purposes (i.e. the disk drives and trash can on Diamond DeskTop). When used, B0 should hold the icon number and W0 should point to the new image data to use.

OVERLAPICON (46) checks if any icons are overlapping a given point on the screen. B0,B1 and B2,B3 represent X,Y coordinate pairs used to form a box around the area that you wish to detect for. After being called, B0 will hold the number of the overlapping icon (1-32) or 0 if no icon overlaps that region. Please note that the icon numbers here are 1 greater than those passed in the previously given routines.

Dialog Boxes

The following six routines relate to the use of Dialog boxes under Diamond. Diamond does not allow Dialog boxes to be stacked (only one can be in use at a given time).

DODIALOG (14) is used to set up a new dialog box. When called B0 should hold the width (in bytes) of the dialog box, and B1 should hold the height (in pixels) of it. The dialog box will automatically be centered for you. W0 points to a list of objects to be drawn inside your Dialog box and W1 points to a list of touch areas for your dialog box. Touch areas are those areas that an activity can occur in. (See sample program DIALOG.M65 and DIALOG.APP)

UPDATEDIALOG (15) allows a dialog box to be refreshed with new objects (or for old objects to be erased). W7 is used to point to a new object list for this routine.

Diamond Develop

EVENTDIALOG (16) detects events in a dialog box. This relies on your list of touch areas and returns which touch area was clicked on as well as the number of clicks issued. B0 returns the touch area clicked on and B1 the number of mouse clicks.

RELEASEDIALOG (17) is used to remove the dialog box from the screen as it handles restoring the screen to its condition prior to DODIALOG. No arguments are needed.

TEXTDIALOG (18) allows for a text area to be opened in a dialog box for keyboard input. B0 holds the length of the string to edit, B1 the starting position in the string, B4 the exit conditions for TEXTDIALOG (0=RETURN key only, 255=RETURN key and when the string has been filled), B5 and B6 correspond to the X,Y position in the dialog box for your text area, W0 the text string to edit, W1 a character filter to filter out unwanted input, and W2 an exit filter. When this routine returns, B0 will indicate what caused the exit (a character or 255 meaning the buffer was full) and B1 will return the ending position in the string.

INVERTTOUCH (49) allows for a touch area to be inverted. While this serves no practical purpose it can be used to 'pretty up' a program. When called, B0 is used to pass the number of the touch area to invert.

Drop-down Menus

The following three routines assist in setting up drop-down menus under Diamond. Diamond supports up to 8 menus with the first one being used by desk accessories. The other seven can contain a maximum of 22 items each.

SETMENU (8) sets up a new menu bar along with its menus. When called, W0 should point to the new menu bar and W1 should point to a table of (up to 8) addresses pointing to each menu structure. (See example program MENU.M65 and MENU.APP)

MENUCHECK (9) allows for the insertion or deletion of a check by individual menu items. When called, B0 should contain the menu number (0-7), B1 the item number (1-22) and B2 the status of the item (0=No check, 8=check). (Note: The 0 and 8 correspond to internal character values so characters OTHER than a check can be placed by the side of a menu item.)

MENUENABLE (10) allows certain menu items to be disabled so that they can no longer be selected. This is helpful in Diamond based applications as it prevents errors. As with MENUCHECK, B0 corresponds to the menu number and B1 the item number. B2 should be 0 for enabled and 1 for disabled. Menu items default to enabled.

Windows

Diamond supports a maximum of four windows that can be open at a single time. Diamond's windows also have many different attributes that can be used.

WINDOPEN (19) is used to open a new window. B0 contains the characteristics that the window is to have (See accompanying table and add bit values), B1 contains the X position of the window on the screen, B2 contains the Y position the window is to have, B3 contains the height of the window, B4 contains the width of the window, and B5 is used to determine if the window is to be buffered or unbuffered (0=Unbuffered, 1=Buffered).

Unbuffered windows can erase the contents of windows behind them and are faster than buffered windows. W0 contains the address of a title bar for the window, W1 contains the address of a subtitle bar for the window (both strings must be terminated with a 255), W2 contains the actual height of the window and W3 contains the actual width of a window. The actual height and width refer to the number of positions the horizontal and vertical sliders can have. Finally, if the window is buffered, W4 and W5 make up a double word pointer to an 8K area

Diamond Develop

of memory used for buffering. Once WINDOPEN is called a window ID # will be returned in W7 (this number will range from 1 to 4). (See sample program WINDOW.M65 and WINDOW.APP)

Available Window Characteristics

Bit Value Characteristic

0	1	Sizer
1	2	Horizontal scroll bar
2	4	Vertical scroll bar
3	8	Drag bar
4	16	Fuller

WINDCLOSE (20) is used to close the top (active) window. No parameters are needed.

WINDMOVE (21) is used to move the top window to a new position on the screen. B0 is used to pass the new X position of the window and B1 is used for the new Y position of the window.

WINDDRAW (22) draws new objects inside of the top window. This routine receives the address of an object list in W7.

WINDCLEAR (42) erases the contents of the top window. No parameters are needed.

WINDGET (43) gets the position of the top window. After WINDGET has been called, B0 and B1 will hold the X,Y position of the window on the DeskTop and W0,W1 will hold the relative position (slider bars) on the window.

WINDSET (50) sets a new actual height and width for the top window. W0 receives the new width and W1 the new height.

WINDTITLE (51) redraws the title bar for the top window. B7 is used to hold a background fill character.

WINDACTIVATE (52) activates a new window and makes it the top window. B0 is the number of the window to activate.

OVERLAPWINDOW (47) checks for windows overlapping a given place on the screen (similar to OVERLAPICON). The arguments passed are identical to those in OVERLAPICON.

Event Handling

EVENT (23) is used to detect events (i.e. a menu item being selected or a window being closed...etc). EVENT receives no arguments and returns any event codes in the system variable EVENTTYPE. The following is an example of how EVENT would be used followed by a table listing all possible return codes for EVENT.

```
LOOP   LDA #23 ;Event Call
        JSR DIAMOND
        LDA EVENTTYPE ;What type of event occurred
        BEQ LOOP ;If 0 then no event occurred
        CMP #1 ;1=Icon event
        BEQ ICON
        CMP #2 ;2=Window event
        BEQ WINDOW
        CMP #3 ;3=Menu item selected
        BEQ MENU
```

Diamond Develop

...
JMP LOOP

Event Type Table

<u>EVENTTYPE</u>	<u>+1</u>	<u>+2</u>	<u>+3</u>	<u>+4</u>
0) No Event Detected				
1) Icon Event	Icon #(1-31), # of Clicks, X position	Y position		
2) Window Event				
	1=Closed			
	2=Fullled			
	3=Dragged			
	4=Resized			
	5=X slider, New position			
	6=Y slider, New position			
	7=Work area, X position, Y position			
	8=New window, window #			
3) Menu Item	Menu #(0-7), Item #(1-22)			
4) Key pressed	ASCII Keycode			
5) Background	# of Clicks, X position, Y position			

Diamond Data Structures

Diamond uses a few basic data structures for fonts, object lists, drop-down menus, and dialog boxes. A data structure is the way in which data is organized for various blocks of data.

Diamond GOS Fonts

<u>Byte</u>	<u>Significance</u>
0	Pixel Height
1-12	12 Character Name
13	Byte width
14	Starting character
15	Number of characters defined
16+	Pixel width of each character (number of entries is equal to the number of characters defined)
...	Character Data

Diamond GOS Mouse cursors

<u>Byte</u>	<u>Significance</u>
0	X Action Spot (0-7)
1	Y Action Spot (0-7)
2-9	Mouse shape data

(See discussion of Mouse related functions for an example)

Diamond GOS Object Lists

0=Image

Word - Address of the image
Byte - X position of the image
Byte - Y position of the image
Byte - Width of the image
Byte - Height of the image

1=Text

Word - Address of the text string
Word - X position of the text string

Diamond Develop

Word - Y position of the text string

Byte - Line spacing

2=Line segment

Byte - X1 position

Byte - Y1 position

Byte - X2 position

Byte - Y2 position

Byte - Color

Byte - Mode

3=Box

Byte - X1 position

Byte - Y1 position

Byte - X2 position

Byte - Y2 position

Byte - Color

Byte - Mode

255=End of object list

Example:

```
OBJLIST
  .BYTE 0 ;Image object
  .WORD IMAGE
  .BYTE 3,3 ;Position of object
  .BYTE 4,16 ;Size of object
  .BYTE 1 ;Text object
  .WORD TEXT ;Addr of text string
  .WORD 24,48 ;Position of text
  .BYTE 8 ;Line spacing (8=normal)
  .BYTE 3 ;Box
  .BYTE 10,10,100,100 ;10,10 to 100,100
  .BYTE 1,0 ;Color 1 in normal mode
  .BYTE 255 ;End of object list
```

Diamond GOS Drop-Down Menus

Diamond GOS menu bars are relatively complex structures as compared with other Diamond GOS data structures. All menu structures consist of a title bar formatted in the following way:

"Menu 1",255(separator),"Menu 2",255,"Menu 3",255,155(End) and up to 8 word sized pointers to each individual menu structure. The first menu bar (which also serves Desk Accessories) consists of a ten byte string (e.g. " INFO "). The other menu structures consist of the following format:

X position under menu bar (0-39), Width (Width of each text entry + 2), # of Items in menu, a 0 for each item (the status of the menu, 0 meaning active), followed by:

```
253,0,"Menu text",253,0,254 for each entry and
253,0,"Menu text",253,0,255 for the last entry
```

The following example should clarify this:

```
MENUBAR
  .BYTE " Desk ",255," File ",255,155
MENUPT
  .WORD DESKMEN
  .WORD FILEMEN
DESKMEN
```

Diamond Develop

```
.BYTE " Info      "  
FILEMENU  
.BYTE 7,9,2,0,0  
.BYTE 253,0," Open  ",253,0,254  
.BYTE 253,0," Close ",253,0,255
```

Diamond GOS Dialog Boxes

Diamond GOS Dialog boxes receive a pointer to an object list as defined and a pointer to a list of 'touch' areas. These touch areas are where Dialog Box Events are detected. The list of touch areas has a simple format and looks like:

```
X1, X2, Y1, Y2, ;First touch area  
X1, X2, Y1, Y2, ;Second touch area  
255           ;End of Touch Area list
```

Diamond Develop

Mouse Drivers

Diamond GOS mouse drivers are files that contain the necessary programming to read a given input device. While this device does not have to be a mouse, we find it convenient to refer to them as mouse drivers. When Diamond GOS first loads up from the Diamond GOS cartridge, the user is set for the default ST Mouse Driver. If the user reconfigures for a different mouse driver this driver is then added to the CONFIG.OS file on the user's Diamond GOS StartUp disk.

The actual mouse driver code consists of a double word pointer that are deposited in MOUSEVEC and DLIMOUSEVEC in the system variable table and up to \$300 (768) bytes of code that will reside at location \$9600 through \$98FF.

Sample source code for our Joystick driver is included (as JOYSTICK.M65). These drivers are then assembled into files (with a DRI extender) and then loaded into our configuration utility for use with the Diamond GOS environment.

Memory Drivers

Memory drivers are similar to mouse drivers as they are also contained in the file CONFIG.OS. If no CONFIG.OS file is supplied, then a default 48K memory driver will be resident which will not allow access to any additional memory (or actually about 256 bytes of it for desk accessories).

A memory driver file consists of a series of vectors to the various memory handling routines (MOVE, MOVETOSCREEN, MOVEFROMSCREEN) and up to \$400 (1024) bytes of code that will reside in memory at \$9000 through \$93FF that actually moves memory. This code is assembled into a file that is then given an extender of DRV and loaded with the CONFIGUR.APP utility as you would a mouse driver.

Sample source code from our 130XE memory driver has been included as MEMORYXE.M65. As with Mouse drivers, these drivers are also loaded into our configuration utility for use with the Diamond GOS environment.

Desk Accessories

Desk Accessories have their inherent power in the fact that they can be accessed from within another program via the left most drop-down menu. Desk Accessories MUST reside in 'extra' memory (under the 48K default driver there is roughly 300 bytes of extra memory for Desk Accessories). Up to six accessories can be resident at one time given adequate memory.

Desk Accessories are loaded when Diamond GOS is started, and they must have an ACC extender for their filename to be recognized. Accessories work by being loaded into extra memory. When called they swap themselves into regular memory and are executed. When the Desk Accessory is done, it must clean up after itself. It will then be swapped back out of memory and control will be returned to the user's application.

The format for a desk accessory is dramatically different from a standard Diamond GOS application (which shares the same file format as Atari DOS II binary files) and is given below:

- 10 bytes for the accessory's name
- 2 bytes for the accessory's length in bytes
- 2 bytes for the base address (the accessory's load address)
- 2 bytes for the accessory's run (execution) address
- The actual accessory program (object code)

Note: A desk accessory must occupy linear memory from its base to its base+(length-1).

Diamond Develop

To aid in the creation of Desk Accessories we have created a Desk Accessory Maker program and a Desk Accessory Skeleton program for use with the Desk Accessory Maker. All this DAM does is takes an assembled desk accessories object code and convert it into desk accessory format. For example, if you assemble SKELETON.M65 into SKELETON.OBJ the OBJ file will not operate as a desk accessory, however, after loading it into our DAM and then saving it as an accessory your desk accessory will now be operable.

Other Included Files

Also included on your Diamond Develop diskette are several other files to aid in the creation of Diamond GOS based applications. They are as follows:

IOMAC.M65 and SYSEQU.M65 - I/O Macros

LIBRARY.M65 - Some very helpful macros including:

- BE addr - Branch if equal (any distance from PC)
- BN addr - Branch if not equal
- BGE addr - Branch if greater than or equal
- BLT addr - Branch if less than
- BGT addr - Branch if greater than
- BLE addr - Branch if less than or equal to
- BER addr - Branch of error (used after a CIO call)
- DINC addr - Double byte increment
- DADD addr,value - Double byte add
- DSUB addr,value - Double byte subtract
- DMOVE value,addr - Double byte move

DMACROS.M65

All of the equates for Diamond functions and memory locations as defined in this manual, and macros for each Diamond function. If you desire to use our macros please make a printout of this to see how to pass arguments, however, using macros is much more limited and less efficient than programming in more conventional means.

FILESEL.M65

As with most graphical operating systems, we have created a standard file selector program for use with your programs. It occupies memory in the \$8000-\$8BFF range. A macro for calling the file selector is also built into this file.

CONFIGUR.M65

This is source code to our Configuration utility to provide developers with source to a complete Diamond GOS application.

DESEG.BAS

Designed for Mac/65 users DESEG will take an object file created with Mac/65 and get rid of all the file segments that it creates. While less advanced users might not understand that, this utility will shrink files assembled with Mac/65 a little and make them load a little faster which is all that one really needs to know.

Appendix I: Function Reference

The following list contains all routines currently available under Diamond GOS in numerical order. Each routine listed contains the routine name, its number, a brief description, and a list of arguments that can be passed to and that are received from the routine. For a more detailed description of each routine and how they interact with each other please refer to the chapter on Diamond's functions.

INIT (0) - Initializes Diamond GOS

Receives:

B0 - Resolution (0=High/1=Low)

Returns:

None

EXIT (1) - Exits the Diamond GOS environment

Receives:

None

Returns:

None

PARAPRNT (2) - Prints a text string using Diamond's styling and font features.

Receives:

B0 - X Offset

B1 - Y Offset

B2 - Line spacing

W1 - X Position

W2 - Y Position

W5 - Address of the text string

Returns:

None

TOGGLEZERO (3) - Swaps Diamond's pseudo-registers with a built in buffer. This routine is used by WINDOPEN and isn't really all that necessary for user applications.

Receives:

None

Returns:

None

INSTALLICON (4) - Installs an icon so that it can be polled for events.

Receives:

B0 - Icon number (0-31)

B1 - X Position (0-39)

B2 - Y Position (0-191)

B3 - Width (1-39)

B4 - Height (1-191)

W0 - Address of the icons bit image

Returns:

None

MOVEICON (5) - Move an existing icon to another position on the screen.

Receives:

B0 - Icon number

B1 - New X Position

Diamond Develop

1,6,1
2,4,1
2,1,1
2,2,1
2,3,4

B2 - New Y Position
Returns:
None

REMOVEICON (6) - Removes an icon that has been installed
Receives:
B0 - Icon number
Returns:
None

SHAPEICON (7) - Changes the shape of an existing icon (used to animate icons e.g. the disk drives on Diamond DeskTop)
Receives:
B0 - Icon number
W0 - Address of new bit image
Returns:
None

SETMENU (8) - Initializes a new menu bar for Diamond
Receives:
W0 - Address of the menu bar
W1 - Address of the menu trees
Returns:
None

MENUCHECK (9) - Insert or remove a check from a menu item (can also place characters other than checks next to menu items)
Receives:
B0 - Menu number (0-7)
B1 - Item number (1-22)
B2 - Character to insert (0=Blank(erase check)/8=Check)
Returns:
None

MENUENABLE (10) - Enable or disable a menu item
Receives:
B0 - Menu number (0-7)
B1 - Item number (1-22)
B2 - 0=Enable/1=Disable
Returns:
None

MOUSEON (11) - Turn the mouse cursor/arrow on
Receives:
None
Returns:
None

MOUSEOFF (12) - Turn the mouse cursor/arrow off
Receives:
None
Returns:
None

Diamond Develop

DEFMOUSE (13) - Defines a new mouse shape

Receives:

W0 - Pointer to mouse shape data

Returns:

None

DODIALOG (14) - Creates a Dialog Box

Receives:

B0 - Width of the Dialog Box

B1 - Height of the Dialog Box

W0 - Address of the Object list for the Dialog Box

W1 - Address of touch areas for the Dialog Box

Returns:

None

UPDATEDIALOG (15) - Updates a Dialog Box (draws a new object list in a Dialog Box and over existing objects)

Receives:

W7 - Address of the Object List

Returns:

None

EVENTDIALOG (16) - Waits for an event to occur in a Dialog Box (please note that this actually **WAITS** for something to happen before returning control to your program)

Receives:

None

Returns:

B0 - Touch area affected

B1 - Number of mouse clicks

RELEASEDIALOG (17) - Terminates usage of the current Dialog Box (must be called after the application is done using the current Dialog Box to clear the screen)

Receives:

None

Returns:

None

TEXTDIALOG (18) - Opens an area in a Dialog Box for text entry. This routine will also filter out invalid input.

Receives:

B0 - Maximum length of the string being edited

B1 - Starting position of the edit cursor (0=beginning)

B4 - Exit conditions (0=Only on C/R,255=When buffer is full)

B5 - X Position of the text area

B6 - Y Position of the text area

W0 - The text string to edit

W1 - Character filter

W2 - Exit filter

Returns:

None

WINDOPEN (19) - Opens a window

Diamond Develop

Receives:

- B0 - Window characteristics
 - 1 = Sizer active
 - 2 = Horizontal scroller active
 - 4 = Vertical scroller active
 - 8 = Drag bar active
 - 16 = Fuller active
- B1 - X Position of the window
- B2 - Y Position of the window
- B3 - Width of the window
- B4 - Height of the window
- B5 - Buffer flag (0=No buffering,1=Buffering)
- W0 - Address of the title
- W1 - Address of the subtitle
- W2 - Actual height of the window
- W3 - Actual width of the window
- W4,W5 - Double word address of the buffer if needed

Returns:

- B7 - The window number used

WINDCLOSE (20) - Closes the active (top) window

Receives:

None

Returns:

None

WINDMOVE (21) - Moves the active (top) window to a new position on the DeskTop

Receives:

- B0 - New X Position
- B1 - New Y Position

Returns:

None

WINDDRAW (22) - Draws objects in a window and updates the window's subtitle

Receives:

- W7 - Address of the object list

Returns:

None

EVENT (23) - Check the Diamond GOS environment to see if any events have occurred. This routine does not WAIT for an event.

Receives:

None

Returns:

None (all info is returned in the system variable EVENTTYPE)

PLOTPOINT (24) - Plots a single point on the screen

Receives:

- B0 - X position
- B1 - Y position
- B2 - Color (0-3)
- B3 - Mode (0=Normal,1=XOR)

Returns:

Diamond Develop

None

PLOTLINE (25) - Plots a line on the screen

Receives:

- B0 - X1 position
- B1 - Y1 position
- B2 - X2 position
- B3 - Y2 position
- B4 - Color
- B5 - Mode

Returns:

None

PLOTBOX (26) - Plots a box on the screen

Receives:

- B0 - X1 position
- B1 - Y1 position
- B2 - X2 position
- B3 - Y2 position
- B4 - Color
- B5 - Mode

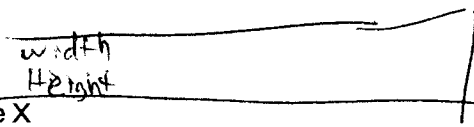
Returns:

None

MOVETOSCREEN (27) - Moves a block image from memory onto the screen

Receives:

- W0,W1 - Address of image data
- B0 - X position
- B1 - Y position
- B2 - Height
- B3 - Width
- B4 - Clipping value X
- B5 - Clipping value Y
- B6 - X offset
- B7 - Y offset



All screen routines have this reversed

Returns:

B7 - \$FF (255) means an error occurred

MOVEFROMSCREEN (28) - Similar to MOVETOSCREEN but instead copies a block image from the screen to memory

Receives:

- W0,W1 - Address of image data
- B0 - X position
- B1 - Y position
- B2 - Height
- B3 - Width
- B4 - Clipping value X
- B5 - Clipping value Y
- B6 - X offset
- B7 - Y offset

Returns:

B7 - \$FF (255) means an error occurred

Diamond Develop

CLEARSCREEN (29) - Clears an area on the screen

Receives:

- W0,W1 - Dummy arguments (usually set to \$2020,\$2020)
- B0 - X position
- B1 - Y position
- B2 - Height
- B3 - Width
- B4 - Clipping value X
- B5 - Clipping value Y
- B6 - X offset
- B7 - Y offset

Returns:

- B7 - \$FF (255) means an error occurred

INVERTSCREEN (30) - Inverts an area on the screen

Receives:

- W0,W1 - Dummy arguments (usually set to \$2020,\$2020)
- B0 - X position
- B1 - Y position
- B2 - Height
- B3 - Width
- B4 - Clipping value X
- B5 - Clipping value Y
- B6 - X offset
- B7 - Y offset

Returns:

- B7 - \$FF (255) means an error occurred

MOVE (31) - Moves a block of memory to another area in memory

Receives:

- W0,W1 - Source address
- W2,W3 - Destination address
- W4 - Number of bytes to move

Returns:

- B7 - \$FF (255) means an error occurred

ZERO (32) - Clears (zeros) a block of memory

Receives:

- W0,W1 - Dummy arguments usually initialized to \$2020,\$2020
- W2,W3 - Address to zero
- W4 - Size of block to zero

Returns:

- None

SWAP (33) - Swap an area of memory with another area of memory

Receives:

- W0,W1 - Address of block 1
- W2,W3 - Address of block 2
- W4 - Number of bytes to swap

Returns:

- None

CURSORON (34) - Turns Diamond's cursor on

Diamond Develop

Receives:
None
Returns:
None

CURSROFF (35) - Turns Diamond's cursor off

Receives:
None
Returns:
None

DEFCURSOR (36) - Defines the cursors shape and blink rate

Receives:
B0 - Bit pattern to use for the cursor (default is \$80)
B1 - Height of the cursor
B2 - Flash rate is 60ths of a second
Returns:
None

MOVECURSOR (37) - Moves Diamond's cursor to a new position on the screen

Receives:
B0 - X position
B1 - Y position
Returns:
None

ADDQUE (38) - Adds a new address to Diamond's VBI queue. Diamond supports more than 1 VBI routine (up to four). New VBI routines can be added via ADDQUE

Receives:
W7 - Address of the new routine
Returns:
B7 - \$FF (255) means no more addresses are available for use

MULTIPLY (39) - Word sized multiply (please note overflow error are not detected)

Receives:
W5 - Multiplicand 1
W6 - Multiplicand 2
Returns:
W7 - Result

DIVIDE (40) - Word sized divide (errors are not detected)

Receives:
W4 - Numerator
W5 - Denominator
Returns:
W6 - Result
W7 - Remainder

SWAPSCREEN (41) - Swaps an area on the screen with a block image in memory.

W0,W1 - Address of image data
B0 - X position
B1 - Y position
B2 - Height

Diamond Develop

B3 - Width
B4 - Clipping value X
B5 - Clipping value Y
B6 - X offset
B7 - Y offset

Returns:

B7 - \$FF (255) means an error occurred

WINDCLEAR (42) - Clears the top (active) window

Receives:

None

Returns:

None

WINDGET (43) - Inquires about the top (active) window's actual position on the screen and relative viewing position in the actual window space

Receives:

None

Returns:

B0 - X position
B1 - Y position
W0 - Relative X position
W1 - Relative Y position

SYSDRAW (44) - System text drawing routine (faster, but less flexible than PARAPRNT)

Receives:

W0 - Address of the text string
B1 - X position
B2 - Y position
B3 - Style of the text

Returns:

None

DRAGBOX (45) - Create a drag box for dragging objects

Receives:

B0 - Initial X position (0-65535)
B1 - Initial Y position
B2 - Width
B3 - Height

Returns:

B0 - New X position
B1 - New Y position
B2 - Mouse's X position
B3 - Mouse's Y position

OVERLAPICON (46) - Inquires about overlapping icons on an area of the screen

Receives:

B0 - X1 position
B1 - Y1 position
B2 - X2 position
B3 - Y2 position

Returns:

B0 - Overlapping icon number (0=None)

Diamond Develop

OVERLAPWINDOW (47) - Inquires about overlapping windows on an area of the screen

Receives:

B0 - X1 position

B1 - Y1 position

B2 - X2 position

B3 - Y2 position

Returns:

B0 - Overlapping window number (0=None)

EXECDESKTOP (48) - Makes a legal exit to the Diamond GOS DeskTop

Receives:

None

Returns:

None

INVERTTOUCH (49) - Invert a touch area in a dialog box

Receives:

B0 - Touch area number

Returns:

None

WINDSET (50) - Sets a new actual width and height for the top (active) window

Receives:

W0 - New actual width

W1 - New actual height

Returns:

None

WINDTITLE (51) - Redraws the title of the active window

Receives:

B7 - Fill character (0=Blanks,10=Shaded)

Returns:

None

WINDACTIVATE (52) - Activates a new window and makes it the top (active) window

Receives:

B0 - New top window number

Returns:

None

BINLOAD (53) - Loads (and executes) a binary file

Receives:

B0 - Diamond disable flag (0=Diamond App,1=Disable Diamond)

W0 - Address of a filename to load

Returns:

None (May execute a program if a run address is given)

MUL40 (54) - Quickly calculate the address of the position of a screen location

Receives:

B6 - X position

B7 - Y position

Returns:

Diamond Develop

W7 - The address of the byte on the screen

FILLSCREEN (55) - Fills an area on the screen with a given bit pattern

Receives:

W0,W1 - Address of image data
W2 - (Low byte only) The fill pattern
B0 - X position
B1 - Y position
B2 - Height
B3 - Width
B4 - Clipping value X
B5 - Clipping value Y
B6 - X offset
B7 - Y offset

Returns:

B7 - \$FF (255) means an error occurred

W0 - File number
W1 - Height
W2 - Width
W3 - Clipping value
W4 - Buffer for data

W7 - Message Box

W0 = Icon

W1 = Text string

W2 = type (0=ok, 1=ok + Cancel)

W3

W4 = 0 for ok, 1 for Cancel

Appendix II: Memory Locations

The following are Diamond GOS's system variables including the label as referenced in our Diamond GOS Macro package, the absolute address of the location, and its purpose.

SCREENX (\$9A80) - The current X position of the mouse (arrow) pointer. Values can range from 0 to 159.

SCREENY (\$9A81) - The current Y position of the mouse (arrow) pointer. Values can range from 0 to 191.

CLICK (\$9A82) - Contains the number of mouse clicks read most recently from the mouse. Values can range from 0 to 3 with the following meanings:

- 0 - No mouse button pressed
- 1 - Mouse button is being dragged (held down)
- 2 - A single click has been detected
- 3 - A double click has been detected

please note that once a single or double click has been detected this location will NOT be cleared automatically so once you read it, find a single or double click has been detected, it is up to your program to clear it.

ACTIVE (\$9A83) - Mouse active status (a 0 here indicates that the mouse is active, a 1 means it is not).

BACKBUF (\$9A84) - A double word (4 bytes) location that points to an area of memory to be used as a screen buffer for drop-down menus and dialog boxes. The default value is \$0000A000 which points to the RAM under the Diamond GOS cartridge. (This must point to an 8K block)

TOTALRAM (\$9A88) - A double word (4 bytes) location containing the size of the memory useable by Diamond including any extra (bank selected) memory. A standard 48K system will read \$00010000 here as this value (64K) includes the ROM in your system.

SYSPTR (\$9A8C) - A double word (4 bytes) that contains the address of the first byte of extra (bank selected) memory.

CLICKTIME (\$9A90) - This location holds the time (in 60ths of a second) that it takes to register a mouse click so as to distinguish it from a drag. Valid values are in the range of 12 to 20 which translates to holding the button down for 1/5th to 1/3rd of a second will generate a mouse click.

PORT (\$9A91) - Contains the number of the (joystick) port that the current input device is being read from. This value is usually a 0 or a 1.

INTERRUPTS (\$9A92) - Status of non-maskable interrupts (NMIs) for the Diamond GOS environment. If interrupts are temporarily disabled while using the Diamond GOS environment, use the value held here to restore them.

EVENTTYPE (\$9A93) - This five byte area is used in conjunction with Diamond GOS's EVENT routine. Values from EVENT are passed back here.

SYSFONT (\$9A98) - This word (2 bytes) area points to the System Font in use for Diamond GOS. The font in use can NOT reside in bank selected memory.

Diamond Develop

NUMFONTS (\$9A9A) - This location was originally intended to hold the number of fonts resident in memory, however, it remains unused at this time.

RESETVEC (\$9A9B) - This double word area (4 bytes) is used to save memory from locations \$A (10) through location \$D (13) when Diamond GOS is initialized. Diamond GOS 'steals' these vectors so it can trap the System Reset button. If the application needs these values they can be found here.

DRIVES (\$9A9F) - These seven bytes are reserved for the drive table that is used by Diamond GOS. Diamond GOS uses drive letters (A-G) instead of drive numbers that are used under DOS. These seven bytes correspond to the seven letters A through G and contain the number of the drive that the given lettered drive is to reference. ASCII values for '0' to '8' are usually found here ('0' indicates an inactive drive).

DEFAULTDRIVE (\$9AA6) - This is the default drive that is used under Diamond GOS for saving the DESKTOP and it will contain a number from 0 to 6 that corresponds to drive A-G.

MOUSEVEC (\$9AA7) - A word sized VBI vector for the mouse driver. This word points to a VBI (vertical blank interrupt) that handles the selected input device.

DLIMOUSEVEC (\$9AA9) - A word sized DLI vector for the mouse driver. This word points to a small routine that is used in detecting/processing motion from the selected input device that runs in a DLI (display list interrupt). At present only the ST Mouse Driver uses this.

XOFFSET (\$9AAB) - The mouse's x offset. The value here usually ranges from 0 to 7 and corresponds to the x position in the mouse shape where action is to be detected. With the arrow mouse pointer, both the XOFFSET and YOFFSET are set at 0 as the very tip of the arrow is to be where actions are detected.

YOFFSET (\$9AAC) - The mouse's y offset. This value typically ranges from 0 to 7 and is used in conjunction with XOFFSET.

MOUSESHAPE (\$9AAD) - This eight (8) byte area holds the mouse's shape data.

EXTRA (\$9AB5) - This double word (4 bytes) holds the address of the first useable byte of extra memory. It (unlike SYSPTR) points above any loaded desk accessories so you can be sure that all memory pointed at and above is free for your application's use.

DESKTOPEXEC (\$9AB9) - A word sized vector that points to the entry point for the Diamond DeskTop.

REZ (\$9ABB) - This location can be used to determine the current resolution that Diamond GOS is being run under. A value of 0 here indicated high resolution (the normal resolution for Diamond GOS), and a 1 here indicates low resolution (color) mode. Color mode is normally not used because windows and text are distorted due to the lower resolution. An example of using Diamond GOS's color mode is the work screen in Diamond Paint.

DOSTYPE (\$9ABC) - This location can be used to determine which DOS (Disk Operating System) Diamond GOS is running under. An 18 here means a form of DOS 2.X, and a 38 indicates that either DOS XE or a form of SpartaDOS is being used (and thus a hierarchical file system/subdirectories).

Diamond Develop

SPARTAFLAG (\$9ABD) - This location is used to distinguish between DOS XE and SpartaDOS when DOSTYPE is set to 38. A 0 here indicates that DOS XE is in use, and a 1 here indicates SpartaDOS is being used.

BANK (\$9ABE) - The current bank number that is active in our Diamond GOS cartridge (The Diamond GOS cartridge consists of 8 8K banks of ROM).

CMDLINE (\$9ABF) - A ³²17 byte command line buffer that is used by Diamond GOS. This is where data is placed from the command line for COM files. The command line is usually terminated with a C/R (\$9B).

BACKFILL (\$9AD0) - A single byte to be used as the background fill pattern whenever Diamond GOS is initialized. The default is a \$55 which corresponds to a %10101010 pattern, or a blue (on 130XE machines).

DELAY (\$9AD1) - Keyboard delay length in 60ths of a second for Diamond GOS keyboard input before auto-repeat becomes active. The default value is 30 which corresponds to 1/2 of a second.

REPEATDELAY (\$9AD2) - Once auto-repeat of a key occurs this is the length of time in between successive repetitions of a key in 60ths of a second. The default value is set at 12 (1/5th of a second).

SCREEN COLOR	\$ ^E 1A73	8B0	709	31651
CLEAR COLOR	\$ 4	015	710	
BACK COLOR	\$ 5	015	712	

SpartaDOS 4.00 ✓
Color

Appendix III: Interfacing Diamond with High-Level Languages

All previous discussion of Diamond has dealt with programming in 6502 assembly (or machine) language. Diamond does not require that you use assembly language to program in. The only requirement is that the accumulator be loaded with a value, registers and memory location be accessible, and that Diamond can be called.

While it is not a requirement that you program Diamond in Assembly language there are some things that just can't be done any other way (as assembly language is the only language that allows you to truly tap the power of the system) and memory will be more limited than it is with assembly language.

Atari Basic

Atari Basic can accomplish this via the USR statement and a routine that has been included to do this for you in the file DEVELOP.BAS. Simply incorporate this into your Basic program and pass the appropriate function values and such to this USR routine and Diamond GOS will do the rest. DEVELOP.BAS demonstrates how to do this with the INIT and EXIT functions.

Another possibility would be to declare all of the symbols used in Diamond in DEVELOP.BAS, however, we elected not to do this as that would consume a lot of precious memory and a large chunk of Basic's variable name table so we recommend using the numeric values supplied in this manual.

ACTION!

To access Diamond simply use the PROC statement (e.g. PROC DIAMOND=\$8E00(BYTE areg) which will allow Diamond to be called via statements of the form DIAMOND(routine #). Registers B0-B7 and W0-W7 can be POKEd and DPOKEd directly.

BASIC XL/XE

Basic XL/XE are both compatible with Atari Basic, however, please note that Basic XE will not work with Diamond in its extended mode due to a memory conflict. At the moment we see no way around this.

C and Pascal

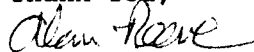
There are several different C and Pascal compilers that have been released for Atari 8-bit computers, however, due to the nature of these languages and the Atari 8-bit computers various sacrifices have been made in their implementations. Also we are unaware of a standard for calling assembly language routines and are unable to supply any more useful information than has been supplied above. We would, however, like to hear from you if you have a C or Pascal compiler that you really like using on your 8-bit machine as we would like to establish one for use with the Diamond GOS environment.

Diamond Develop

Conclusion

We hope you enjoy developing software for the Diamond Graphical Operating System as we see a strong future for this program and Atari 8-bit computers as a whole if we get some strong support from independent developers. If you should design a program and would like us to make it available to other GOS owners please feel free to send it to us. If you would like us to evaluate your program for possible commercial distribution we'd love to see that too.

Thank You,



Alan Reeve

REEVE Software OnLine

GENie users can contact us online in the Atari 8-bit Roundtable. Our message library in the bulletin board in category #14 and our software library is #26. Our EMail is REEVE.SOFT and we check in roughly once a day to answer questions and such.

On CompuServe we can be reached at user ID 71521,2200. We do not check in quite as frequently there so please be patient if your questions are not answered overnight.

Include character set page.