Chapter 24
Miscellaneous Amiga Support

What is supported and why?

You may be wondering how we decided which Amiga functions we would define JForth words for and which not. Our primary goal is to provide the necessary tools for accessing any Amiga functions. The CALL system and the Structure referencing tools provide this. These are theoretically all you need to access the Amiga libraries. This is why some functions are not directly supported. As for why some function are supported, there are four main possible reasons. The first reason might be that a function is so important that almost everyone will want it. The second is that the function might be a little tricky to implement so we will save you the pain, like the polygon or IFF code. The third reason is if it is a function normally present in 'C' but not part of the Amiga libraries, eg. CreateStdIO(). The final reason would be if we needed a function to write one of the demos or applications and then included it in a support file for all to use.

The routines that follow all pass addresses as relative addresses. Words that end in a '()' suffix map are simply calls to the Amiga library functions. Most of these calls are described in detail in the ROM Kernal Manual.

File JU:GRAPH_SUPPORT

These routines support various Graphics and Intuition Library calls.

`ALLOC.BITMAP ( bdepth bwidth bheight -- bitmap | false )`

Allocate a BitMap structure with bitplanes based on the input. Calls InitBitMap(). This BitMap must eventually be freed using FREE.BITMAP.

`ALLOC.RASTPORT ( -- rastport | 0 , allocate and initialize)`

`ALLOC.SHADOW ( bdepth bwidth bheight -- bitmap | false )`

Allocates a special kind of bitmap that has only one plane allocated for each plane position. This is used for building shadow masks for transparent Blit operations. Must be freed using FREE.SHADOW.

`ALLOCRASTER() ( x y -- address )`

`BITMAP>WH ( bitmap -- width height , extract information )`

`BltBitMapRastPort() ( bitmap x y rp x y w h minterm -- )`

`BltClear() ( memblock bytecount flags -- )`

`BMPLANE[] ( index bitmap -- &plane-ptr )`

This points to the position in the BitMap that holds the address of the plane. It is not the plane address itself. Do a @ >REL to get the plane address.Used when allocating and assigning planes both.

`CLEAR.BITMAP ( bitmap -- , set planes to zero )`

Clear all the bit planes in a BitMap to zero using BltClear().

CLEAR.BITPLANE (plane# bitmap -- , clear plane of bitmap)

ClipBlit() (src_rp x y dest_rp x y w h minterm --)

CLOSEFONT() (font --)

COPY.PLANES (src_bm dest_bm -- , copy planes into other bitmap)

This copies the pointers to the planes of one BitMap to another BitMap. This can be useful for double buffering. Be careful you don't free these planes twice!

CTABLE>RGB (index ctable -- r g b , extract colors)

Extract Red, Green, and Blue values from a colortable. A colortable is packed 0RGB in 16 bits.

FREE.BITMAP (bmap -- , free planes then the Bitmap)

FREE.SHADOW (bmap -- , free plane then the bitmap)

For freeing Shadow mask allocated by ALLOC.SHADOW.

FREE.VIEW (view -- , free CPR lists and deallocate)

FreeCprList() (CPRList --)

FREERASTER() (address x y --)

FreeVPortCopLists() (viewport --)

INITBITMAP() (bitmap depth width height --)

INITRASTPORT() (RastPort --)

INITTMPRAS() (tmpras buffer size --)

INITVIEW() (view --)

INITVPORT() (viewport --)

LOADRGB4() (viewport colortable n -- , load color palette)

LOADVIEW() (view --)

MAKEVPORT() (view viewport --)

MRGCOP() (view --)

PACK.RGB (r g b -- ctable-entry)

Pack the 4 bit Red, Green, and Blue values into 16 bits organized as 0RGB.

RASSIZE() (height width -- size_in_bytes)

REMAKE.SCREEN (screen -- , rebuild Amiga display system)

RGB>CTABLE (r g b index ctable -- , store colors)

Opposite of CTABLE>RGB. Uses PACK.RGB

ROTATE.PLANES (bitmap -- , rotate planes for effect)

An interesting color shifting effect can be obtained if you rotate the order of the plane pointers in a Bitmap. This is very fast since you are only changing pointers. You must rotate as many times as there are planes to get back to normal.

SCALE.COLOR (color scalar -- color' , 0-16 scale)

Scale a 4 bit color value where the scale ranges from 0 to 16 where 16 is equivalent to 1.0.

```
7 16 scale.color  ( get 7 back )
7 8  scale.color  ( get 3 back )
```

SCALE.CTABLE (ctable1 ctable2 many scalar -- , scale c1 into c2)

Copy a colortable into another by unpacking the RGB values, scaling them, then repacking them back into the new colortable. Useful for fading in and out a picture.

SCALE.RGB (r g b scalar -- r' g' b')

SetRast() (rastport pen --)

SWAP.PLANES (bmap1 bmap2 -- , swap planes between bitmaps)

SWITCH.SCREEN (rastport screen -- , switch double buffer)

If the RastPort is used for drawing and the screen for displaying, you can create a double buffered image by drawing, then calling SWITCH.SCREEN.

WAIT.FRAMES (N -- , wait for N blanking intervals)

Loop on WaitTOF()

WaitTOF() (-- , wait till next vertical blank)

Font Support

See the demo JD:DEMO_FONT for an example of the use of these routines.

GET.FONT (0name ysize -- font | NULL)

Fills a text attribute structure with the necessary data and gets the disk based font. This font can then be passed to GR.FONT! for use with the graphics toolkit.

OPENDISKFONT() (textattr -- font | NULL)

OPENFONT() (textattr -- font | NULL)

SETFONT() (rp font --)

File JU:GADGET_SUPPORT

The Amiga Gadget system provides an flexible way to control your programs. We have provided a set of routines for initializing the common Gadget structures to typical values. You can then modify the values to suit your particular application. See the demo JU:DEMO_GADGET for an example of their use.

BOOLEAN.SETUP (x y width height gadget --)

BORDER.SETUP (width height border --)

BOX.SETUP (width height addr -- , set 5 points for box)

Used when creating a simple Border.

CHECKBOX.SETUP (x y width height gadget --)

INTGADGET.SETUP (x y width height gadget --)

ITEXT.SETUP (text0 intuitext -- , Set defaults.)

MENUBUTTON.SETUP (x y width height gadget --)

PROPGADGET.SETUP (x y width height gadget --)

PROPINFO.SETUP (hpot vpot hbody vbody propinfo --)

REFRESHGADGETS() (gadgets ptr req --)

STRINGGADGET.SETUP (x y width height gadget --)

STRINGINFO.SETUP (buffer maxchars stringinfo --)

File JU:POLYGON for Area Fill

To draw polygons you must first set up a temporary RastPort for creating a mask. GR.AREA.INIT will create one based on the current RastPort whose absolute address is in GR-CURRPORT. Then you can start a polygon by calling GR.AREA.MOVE followed by some number of calls to GR.AREA.DRAW then finally calling GR.AREA.END to draw the polygon. When you are all done drawing all your polygons, please call GR.AREA.TERM to deallocate the temporary RastPort. The system is set up with a limit of 100 points per polygon. Change the source code if you need more.

GR.AREA.INIT (-- , setup Temporary RastPort for polygons)

GR.AREA.TERM (-- , deallocate temporary RastPort)

GR.AREADRAW (x y --)

GR.AREAEND (--)

GR.AREAMOVE (x y --)

GR.TRIANGLE (x1 y1 x2 y2 x3 y3 --)

INITAREA() (areainfo areabuffer count --)

File JU:SCREEN_SUPPORT

The file JD:DEMO_HAM is a good example of these calls. Please see the Intuition manual for a detailed description of these calls.

BITMAP>SCREEN (bitmap viewmode -- screen | NULL)

Opens a screen that uses the given bitmap as its CUSTOMBITMAP.

CLOSESCREEN() (screen --)

MAKESCREEN() (screen -- , similar to MakeVport)

MOVESCREEN() (screen deltax deltay --)

A positive deltay will move the screen down. Deltax is ignored.

NEWSCREEN.SETUP (NewScreen -- , Set default values)

You can call this routine first then override the default values by setting them to your own.

OPENSCREEN() (NewScreen -- screen)

REMAKEDISPLAY() (-- , remake screens and COPR list)

RETHINKDISPLAY() (-- , reconstruct COPR list): SCREEN>VIEW (ascreen -- aview | NULL)

Moves screen to front then allocates a view based on the current display. This view can be used with LoadView() to create double buffered displays. Returns NULL (0) if the view could not be allocated. Free the view using FREE.VIEW

SCREEN>BACKWINDOW (screen -- window | NULL)

Creates a BACKDROP window to fit a screen. Sets the FLAGS and IDCMPFLAGS to the values in S>B_FLAGS and S>B_IDCMP_FLAGS. The default values are shown below.

```
BACKDROP BORDERLESS | WINDOWCLOSE | REPORTMOUSE |
-> S>B_FLAGS


CLOSEWINDOW MOUSEBUTTONS |
VANILLAKEY | MENUPICK | GADGETUP | GADGETDOWN |
-> S>B_IDCMPFLAGS
```

SCREENTOBACK() (screen -- , push to back)

SCREENTOFRONT() (screen -- , bring to front)

SHOWTITLE() (screen flag -- , SHow title bar or not.)

Exec Library Support

These words support most of the calls in the Exec Library. They can be found in the file JU:EXEC_SUPPORT. All addresses are passed as JForth relative addresses. Please see the ROM Kernal documentation for a description of these calls.

ADDPORT() (port --)

ALLOCSIGNAL() (requested_signal# -- allocated_signal#)

CHECKIO() (ioreqblk -- ioreqblk | 0)

CLOSEDEVICE() (IORequest --)

DOIO() (ioreqblk -- error , Do I/O)

FINDPORT() (0$portname -- task)

The portname is passed as a NUL terminated string which must be generated using 0" inside of a colon definition.

FINDTASK() (0$taskname -- task)

FREESIGNAL() (signal# -- , Free previously allocated signal.)

GETMSG() (port -- message , Get message.)

OPENDEVICE() (0$devicename unit# IORequest flags -- error)

PUTMSG() (port message -- , Put message.)

REMPORT() (port --)

ReplyMsg() (message --)

SENDIO() (ioreqblk -- error)

WAITIO() (ioreqblk -- error)

WAITPORT() (port -- message)

Other Exec words - CreatePort() AbortIO() etc.

The following words are commonly used but are not part of the Exec Library. They are traditionally supplied as 'C' macros or are listed as 'C' source code in varous texts. They have been converted to JForth code for your convenience. You should always check the return value to make sure it is non-zero. Unless otherwise specified, these routines can be found in JU:EXEC_SUPPORT.

ABORTIO() (IOReqBlock -- result , abort an IO request)

This is a special call directly off of a Device structure. It returns a result but it is meaningless so you can drop it. We found out about this when it was too late to change it. This is defined in JU:DEVICE-CALLS. The above is also true of BeginIO().

BEGINIO() (IOReqBlock -- result , start an IO request)

See ABORTIO() definition.

CREATEEXTIO() (ioreplyport size -- ioreq | 0)

Allocate an extended IO Request Block that will use an existing IOReplyPort.

CREATEPORT() (0$name priority -- msgport | 0)

Allocate and add a new port to the system.

CREATESTDIO() (ioreplyport -- ioreq | 0)

DELETEEXTIO() (ioext size --)

DELETEPORT() (port -- , Delete message port)

DELETESTDIO() (ioreplyport --)

PORT.SETUP (0$name priority signal msgport -- , Set values.)

Amiga Linked List Tools

These routines can be found in the file JU:EXEC_LISTS. They are Forth routines based on 'C' macros. They are handy for examining Amiga system lists, or for creating your own lists.

DUMP.LIST (list -- , dump nodes of list)

DUMP.NODE (node -- , dump contents)

FORBID() (-- , forbid other tasks)

If you are looking at system lists, you should forbid other tasks from changing them. Call PERMIT() when done.

LIST.ADDTAIL() (list node -- , add node to end of list)

LIST.FIRST() (list -- first_node , get first node in list)

LIST.IFEMPTY() (list -- flag , true if empty)

NEWLIST() (list -- , Initialize list header)

Set the pointers in a list header to point to itself. This signifies an empty list.

NODE.AFTER (node1 node0 -- , connect node1 after node0)

NODE.INIT (type priority 0name node -- , initialize node)

NODE.SUCC() (node -- succeeding_node)

PERMIT() (-- , permit other tasks after FORBID())

ANSI Text and Cursor Control

These commands allow the control of text and the cursor in a console window. They send special control characters to the current output which are interpreted by the console device. These commands are used extensively by the Command Line History system. See also the file JU:COLORDUMP for more examples. Please see the ROM Kernal Manual for more information.

Note: The ANSI referred to here is the ANSI standard for terminal control characters, not the upcoming ANSI Forth standard. ANSI stands for American National Standards Institute. They standardize everything from languages to lightbulb sockets.

.DECIMAL (n -- , output decimal number without spaces)

This is used to generate formatted ANSI commands.

>STYLE (n -- , select graphic rendition)

This is used by ITALICS and other ANSI words to select a style.

ANSI.1C (char -- , send CSI and one char)

ANSI.BACKWARDS (N -- , move cursor backwards)

ANSI.DELETE (N -- , delete character under cursor)

ANSI.DOWN (N -- , move cursor down)

ANSI.ERASE.EOL (-- , erase to end of line)

ANSI.FORWARDS (N -- , move cursor forwards

ANSI.INSERT (N -- , insert character)

ANSI.NC (N char -- , send a number and a character sequence)

This is used to build special custom ANSI commands.

`ANSI.PARSE.SKR  ( -- key-code, get packed key sequence)`

If KEY returns a 155 decimal character, call this to parse the remaining characters. It will eat keys and return a FKEY index equal to the following. You can then use this in a CASE statement. Remember that if history is on it will eat these special keys so your program will never see them. Call HISTORY.OFF if you want to handle these characters yourself.

```
0 = error
1 = function key 1, 11 = shift 1,
2 = function key 2, 12 = shift 2, etc.
21 = cursor-up, 22 = cursor-down,
23 = cursor-right, 24 = cursor-left,
25 = shift-cursor-up, 26,27,28 (shifted v < >)
29 = help
```

`ANSI.UP ( N -- , move cursor up )`

`B:0 B:1 B:2 B:3 ( -- , set text background color, see F:0 )`

`BOLD       ( -- , make characters bold )`

`F:0 F:1 F:2 F:3 ( -- , set text foreground color )`

These words will change the text color. If history is interfering with this color, enter:

```
HIGHLIGHT-INPUT OFF
F:3  ( for fun )
B:2
```

`GOTOXY   ( x y -- , move text cursor to column x, row y )`

`INVERSE    ( -- , style 7 )`

`ITALIC     ( -- , style 3 )`

`PLAIN      ( -- , style 0 )`

`UNDERSCORE ( -- , style 4 )`

Amiga DOS 2.0 Support

The ".j" include files were translated from Commodores AmigaDOS 2.0 ".h" files. The conversion from 'C' to Forth was done using H2J which is in JA:H2J.F

JForth makes the full field of AmigaDOS 2.0 features available to the programmer. It is the programmer who must ensure that AmigaDOS functions and capabilities are not invoked under earlier versions of the operating system.

CHECK **2.0FEATURES?** BEFORE CALLING AMIGADOS 2.0-SPECIFIC FEATURES AND FUNCTIONS!

Identifying the Workbench Version

`2.0FEATURES?  ( -- flag )`

Returns a TRUE flag if running under AmigaDOS 2.0 or later. The programmer should ensure his program is operating under 2.0 before using its services.

JU:ASL_SUPPORT

ASL - the Application Support Library

ASL is a new library provided in Amiga DOS 2.0. It provides facilities like file and font requesters. We urge programmers to use these requesters in their applications because they provide the user with a familiar interface. One problem, however, is that if the user is running AmigaDOS 1.3, these cannot be used. We suggest that you either require AmigaDOS 2.0 OR you should use the requester in JREQ:FileRequester.f when running under AmigaDOS 1.3.

Here are some direct calls to the ASL library. You must call ASL? before calling these routines.

`AllocASLRequest() ( type ptags -- request )`

`AllocFileRequest() ( -- request )`

`ASLRequest() ( request ptags -- result )`

`FreeASLRequest() ( request -- )`

`RequestFile() ( request -- result )`

ASL Forth Utilities

We have provided some words that use the ASL library as a convenience and as examples. These require AmigaDOS V2.0 or higher.

`ASL.FILENAME>PAD ( file-request -- , copy full name to pad )`

This can be called after using the File Requester. It will concatenate the directory names and file names into a useable path name. The result will be left on the PAD. It is used internally by ASL.REQUEST.FILE

`ASL.REQUEST.FILE ( 0prompt -- $name true | false )`

Open the file requester with the given prompt. If the user selects a file, return its name and a true. If the user cancels the operation, return false. The filename will be left on the pad. See also the file requesters in JARP: and JREQ:.

```
: TEST.ASL ( -- )
   0" Select a file." ASL.REQUEST.FILE
   IF  COUNT TYPE CR
   THEN
;
```

`ASL.REQUEST.FONT ( 0prompt -- font true | false )`

Open the font requester with the given prompt. If the user selects a font, it opens it and returns a font pointer and a true. If the user cancels the operation, return false. This routine opens the diskfont library and calls OpenDiskFont(). When you are done with the font, you should pass it to CloseDiskFont(). You can use this font by calling Intuition's SetFont(). See also GR.FONT! in JU:AMIGA_GRAPH.

Tag Lists

TagLists are a new data structure in AmigaDOS 2.0. They are used to passed multiple attributes to a routine. The attribute identifier and a value are passed as pairs in a "list" that is terminated with a NUL. These routines simplify the creation and use of TagLists. Taglist support is in JU:TAGLIST.

`TAGLIST{  ( taglist -- )`

Start the definition of a taglists. You must decide where to put the taglist. You can put it in TAG-PAD, or in a memory area that you allocate.

`}TAGLIST ( -- taglist )`

Finish the definition of a taglist. This returns the address you passed to TAGLIST{ as a convenience.

TAG-PAD (-- addr)

A place to put a taglist. The maximum number of tags it will hold is 16. Do NOT exceed 16 tags. If you need more tags, put them on the normal PAD or allocate memory. This is used by ASL.REQUEST.FILE.

TAG, (value -- , add to current taglist)

This is used by +TAG to add a single value to a taglist.

+TAG (attr value --)

Add an attribute/value pair to a taglist.

+TAG.ABS (attr rel-addr --)

Addresses in a taglist must be in absolute form for the Amiga. Use this word when adding addresses to a taglist. If will convert non-zero values to absolute using IF>ABS.

TAGLIST.DUMP (taglist -- , dump taglist maximum 100 long)

Dumps a taglist and shows each value as a number and as an address.

Here is an example of using these words to create and display a taglist. This taglist uses some of the tags for OpenWindowTagList().

```
INCLUDE? TAGLIST{  JU:TAGLIST
: MAKE.TAGLIST ( -- taglist )
   PAD       \ arbitrarily put it on PAD
   TAGLIST{  \ begin definition
      WA_HEIGHT  200  +TAG  \ set window height
\ now specify a custom screen ADDRESS
      WA_CUSTOM_SCREEN MY-SCREEN @ +TAG.ABS
   }TAGLIST
   dup taglist.dump
;
```