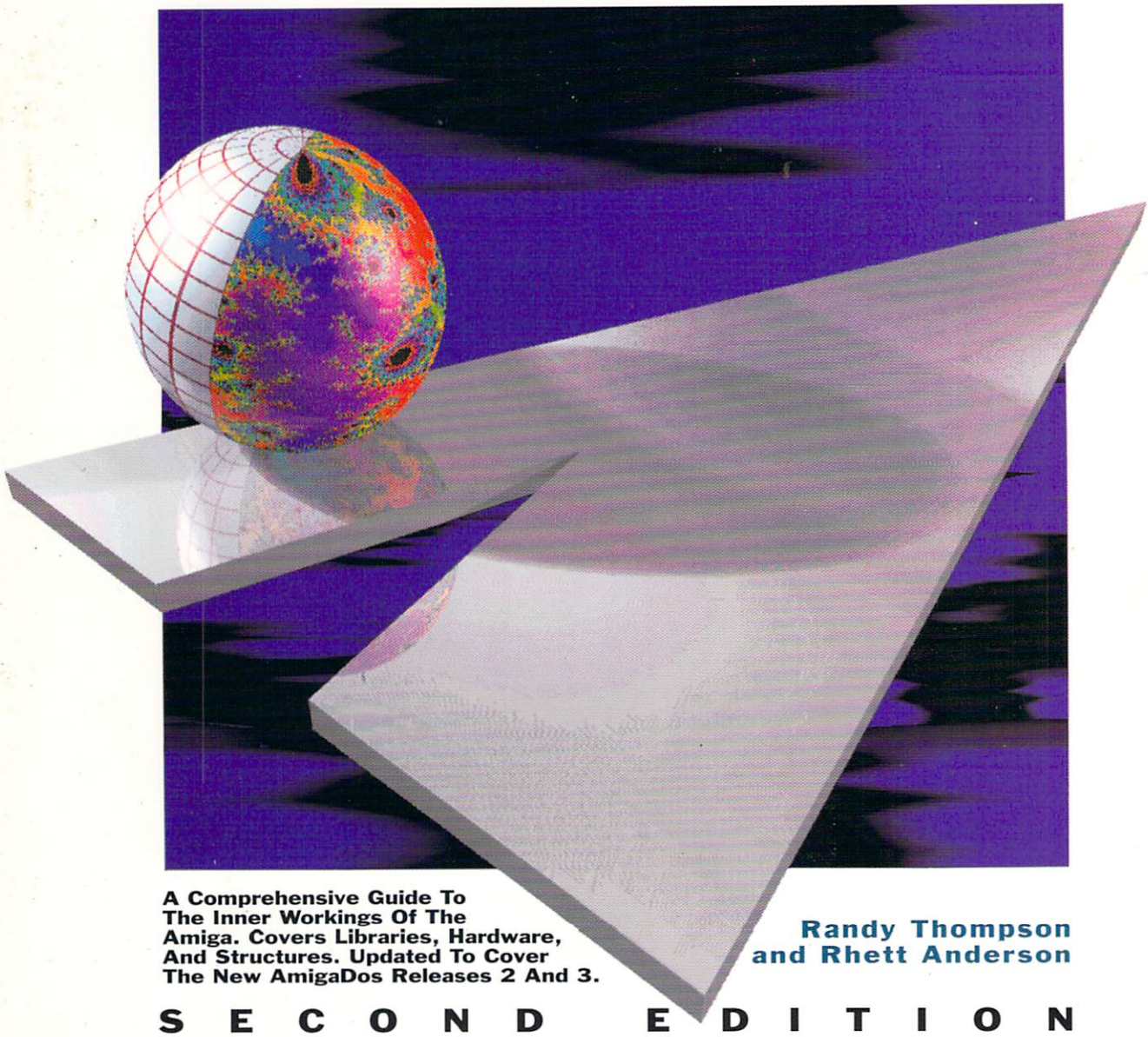
MAPPING THE AMIGA



A Comprehensive Guide To The Inner Workings Of The Amiga. Covers Libraries, Hardware, And Structures. Updated To Cover The New AmigaDos Releases 2 And 3.

Randy Thompson
and Rhett Anderson

SECOND EDITION

Mapping the Amiga®

Second Edition

Rhett Anderson and Randy Thompson

COMPUTE Books
Greensboro, North Carolina

Cover design: Lee Noel
Editor: Stephen Levy
Tysetting: Terry Cash

Printed in the United States of America

10 9 8 7 6 5 4 3 2

ISBN 0-87455-267-2

COMPUTE Books, 324 W. Wendover Ave., Suite 200, Greensboro, North Carolina 27408, is a General Media company and is not associated with any manufacturer of personal computers. Amiga is a registered trademark of Commodore-Amiga, Inc.

Contents

Preface

There are so many programming books for the Amiga. Why should there be another one?

Frankly, we wrote this book for ourselves. When we sit down to program, we find no single book is enough. The typical Amiga programmer may have Commodore's official reference manuals (which, as of this writing, still don't cover Revision 3.0), the complete set of *Amiga Transactor* magazines, and even several printouts of downloaded programming information and source code, all scattered across desks, chairs, and, of course, the floor. As you know, programming the Amiga can be a daunting task. You might spend an hour just tracking down the right source. What is the solution to this problem?

We turned for inspiration to two books that helped us in the past: *Mapping the Atari* by Ian Chadwick (1983, COMPUTE! Books) and *Mapping the Commodore 64* by Sheldon Leemon (1984, COMPUTE! Books). These books had it all—explanations of how the respective machines organized memory, detailed listings of what various memory locations were and how to use them, descriptions of input/output registers, and discussions of the functions of various ROM routines.

These books were so detailed a programmer could literally write a program without putting fingerprints on a single other book.

But of course you can't really do the same thing for the Amiga. The above mentioned books were slightly over 200 pages. This book is more than twice as large and comes nowhere near saying everything that can be said about the Amiga. If the Apple II's built-in software and hardware is an apartment, the Commodore 64's is a townhouse, and the IBM PC's is a modest home in the suburbs, then the Amiga's is a Manhattan City block.

And the problem is not just a matter of scale. Unlike earlier computers, the Amiga doesn't just set aside areas of memory for this or that purpose. Instead, the Amiga has libraries of routines, which can show up virtually anywhere in the RAM space of the machine. These routines aren't even all in ROM; some are loaded from disk. The screen is not in a fixed location. Commodore even threatens to change the locations of the memory-mapped custom chips. In fact, with the advent of the new AGA chips and the upcoming AAA chips, Commodore has already moved and/or modified many of the hardware registers.

The Amiga doesn't go through all these contortions to be contrary. Instead, this level of indirection provides the Amiga with prodigious flexibility. Libraries are easily updated and added. Peripherals and input devices are easily accommodated.

Aware of the problems of scale and flexibility, we set out to accomplish a different and (somewhat) more reasonable task: to enable a programmer to use our book as a reference work that answers 90 percent of his or her questions. This task took two of us a year, and that was just for the book's first edition. In order to add the daunting amount of new Revision 3.0 information, we couldn't do it alone.

Kudos

Nathan Dwyer and Peter Heinrich provided invaluable help by compiling the data for most of Chapter 1—without these two, this revised edition of *Mapping the Amiga* would not have been possible. A special thanks also goes to Sheldon Leemon for his many suggestions and observations, and to Stephen Levy, our book editor, for his patience and persistence in ensuring that this vital revision made it to print.

How to Use This Book

First, we suggest a little window shopping. Browse among the chapters. Ever wonder how the blitter works? Curious about the HAM video mode? Cruise the Hardware chapter. Want to know how the Amiga organizes its data? Want to know how to use the Amiga's built-in functions? The Structures and Libraries chapters tell you. Becoming familiar with the structure and content of the book will pay off later.

The majority of this book consists of tables and charts—a compilation of vital programming information organized in understandable and usable formats. Much of this information can't be found in any other single place. For example, where else can you find a complete alphabetical listing of library functions that gives the syntax of each function in C and machine language?

Each major section of the book begins with an introduction that explains how to use the information that follows. So even if you're a beginner, you'll find this book a great way to learn the Amiga's inner workings.

If you already have a stable of Amiga programming books, notice how the layouts of those books compare with ours. For instance, some books group functions by the library in which they're found; *Mapping the Amiga* orders the functions alphabetically. Knowing this can save you some time and help you learn how to use *Mapping* in conjunction with your other programming books.

If you don't have many other programming books, we'd like to suggest a few. Commodore's set of reference manuals (published by Addison Wesley, except for the

AmigaDOS book, which is published by Bantam) are invaluable. Be sure to get the latest set; new examples and tables have been included in every update so far. Sybex sells three excellent books: Eugene Mortimore's two-volume *Amiga Programmer's Handbook* and Rob Peck's *Programming the Amiga*. Carl Sassenrath publishes his own *Guru's Guide* to the Commodore Amiga book, which is a must-have for anyone wanting to write interrupt code. Compute Books also has many titles. We often turn to *COMPUTE!'s Amiga Programming Guide* and Sheldon Leemon's *Inside Amiga Graphics*. Of course, you'll also want a manual for the language that you're using.

A Book You'll Use

We hope this book is as useful to you as it is to us—not a book you want on your bookshelf, one you want on your desk, easily accessible and face open.

As Bill Wilkinson said in the introduction to *Mapping the Atari*, good luck and happy mapping.

Chapter 1
Library Functions

The Amiga is an incredibly capable machine. It has so much to offer, from digitized sounds and dazzling color graphics to a window-based user interface and a true multitasking operating system. Library functions are the key to accessing these features.

The Amiga's library functions are similar to the Commodore 64's ROM Kernal subroutines, the IBM PC's BIOS interrupt calls, and the Apple Macintosh's Toolbox procedures—they are a collection of routines that allow you to control almost every aspect of your computer. They're called *functions* because the majority of them were originally written in the C programming language. In C, all user-defined commands are referred to as functions.

Linked Libraries

There are really two types of libraries on the Amiga: *linked libraries* and *shared libraries*.

Linked libraries are a collection of external functions that you make part of your program. When you link your program with Blink, each linked library function you use is brought in from one of the specified .lib files and stuffed into your program. The SAS/C file sc.lib and the Manx C file c.lib are both examples of linked libaries.

The functions offered by a linked library are dependent upon your compiler or assembler, except for amiga.lib. The amiga.lib linked library offers common functions such as printf( ), atoi( ), and BeginIO( ), although many of these functions are often replaced by your C compiler's linked library. Linked libraries are explained in the manual that came with your compiler or assembler. For that reason, only the official Commodore Amiga shared libraries are explained in this chapter.

Shared Libraries

As the name implies, a *shared library* is accessible to all Amiga programs. When you use a shared library function, its code does not become part of your program on disk; shared libraries are located in ROM (writable control memory on the venerable Amiga 1000) or loaded into RAM from the LIBS: directory of your Workbench disk.

Shared libraries are more memory conservative than linked libraries. Every

program that accesses a function from a linked library must contain its own copy of that function. Only one copy of a shared library function ever needs to be in memory because all programs have access to it. When someone refers to an Amiga library, they're almost invariably talking about a shared library. The same is true of this book.

Under Kickstart/Workbench 1.3 there are over 15 libraries. Revision 2.0 added another six, and Revision 3.0 added four more. In addition to expanding the number of libraries available to the Amiga programmer, both Revision 2.0 and 3.0 added many new functions to existing libraries and enhanced several more. Each library controls a different part of the Amiga. For graphics programming, for example, you use the graphics.library; for creating windows, requesters, and pull-down menus, you use the intuition.library; for disk access, you use the dos.library.

If you are still using version 1.2 or 1.3 of the Amiga's operating system, seriously consider upgrading. Not only will your programs run better, but, with all the new easier-to-use libraries and functions available to you, it should take you less time to write your programs.

Opening and Closing Libraries

A library must be opened before you can access its functions. Like your local Seven-Eleven store, the Exec library is always open. This works out rather well considering OpenLibrary( ), the one function required to open other libraries, is contained in Exec. When you link with startup code such as SAS/C's c.o, the DOS library is opened for you as well.

You open a library using the aptly named OpenLibrary( ) function. OpenLibrary( ) expects two arguments: the library name and the library version number. (See Table 1-2 for a complete list of available libraries.) The sample code below shows the C and machine language syntax for using the OpenLibrary( ) function.

In C:

```
libBase = OpenLibrary(libraryName,version);
```

In machine language:

```
move.l   ExecBase,a6       ;Pointer to ExecBase
lea      libraryName,a1    ;Pointer to the library's name in a1
moveq    #version,d0       ;Version number goes in d0
jsr      OpenLibrary(a6)   ;Open library and return libBase in d0
move.l   d0,libBase        ;Save libary base pointer
```

where libBase is the name of the library's base pointer, libraryName is the name of the library, and version is the lowest acceptable version of the library you require. For Revision 2.0, for example, you should use a version number of 36. The following table explains all the possible version numbers you can use:

Table 1-1. Library Version Numbers

Version Number	Operating System
0	Any version
30	OS 1.0
31	NTSC OS 1.1
32	PAL OS 1.1
33	OS 1.2 (oldest version still supported)
34	OS 1.3
36	OS 2.0
39	OS 3.0

If you specify an operating system version that is higher than what is available on your Amiga, OpenLibrary( ) will fail and return a NULL (zero) value. If the library is opened successfully, the base address of the library is returned in libBase.

Table 1-2 shows the library base pointer names (libBase) and their corresponding library names (libraryName) that you should use when opening a library. In machine language, you can use practically any name you choose, as long as you define an area in memory to store the pointer and label that location using the name you select. By convention, it's recommended you use the library base pointer name provided in Table 1-2 and precede it with an underscore character, as in the name _DOSBase. In SAS/C, you must use the names given below. If you don't, your program will compile and link correctly, and then happily crash when it is run.

Table 1-2. Library Names

Library Name	Library Base Pointer Name
amigaguide.library**	AmigaGuideBase
asl.library*	AslBase
bullet.library**	BulletBase
commodities.library*	CxBase
datatypes.library**	DataTypesBase
diskfont.library	DiskfontBase
dos.library	DOSBase
exec.library	ExecBase (always memory location 4)
expansion.library	ExpansionBase
gadtools.library*	GadToolsBase

Library Name	Library Base Pointer Name
graphics.library	GfxBase
icon.library	IconBase
iffparse.library*	IFFParseBase
intuition.library	IntuitionBase
keymap.library	KeymapBase
layers.library	LayersBase
locale.library**	LocaleBase
mathffp.library	MathBase
mathieeedoubbas.library	MathIeeeDoubBasBase
mathieeedoubtrans.library	MathIeeeDoubTransBase
mathieeesingbas.library	MathIeeeSingBasBase
mathieeesingtrans.library	MathIeeeSingTransBase
mathtrans.library	MathTransBase
rexxsyslib.library*	RexxSysBase
translator.library	TranslatorBase
utility.library*	UtilityBase
workbench.library	WorkbenchBase

* new with Revision 2.0
** new with Revision 3.0

When your program is finished, you must be sure to close all of the libraries you opened. To accomplish this you use yet another aptly named Exec function, CloseLibrary( ). The only argument that the CloseLibrary( ) function requires is the base address of the library you wish to close. This is the same base address that was returned by OpenLibrary( ). Here are some examples on using CloseLibrary( ):

In C:

```
CloseLibrary(libBase);
```

In machine language:

```
move.l   libBase,a1        ;libBase goes in register a1
move.l   ExecBase,a6       ;Pointer to ExecBase goes in a6
jsr      CloseLibrary(a6)  ;Close the library
```

If you neglect to close a library, you run the risk of wasting memory since the system will not know if it can throw out an unused, RAM-based library.

Calling Functions

The end of this chapter contains an alphabetical listing of every public Amiga library function available—an impressive collection. Each table entry gives the function's name (followed by a parenthetical note, if the function is new under Revision 2.0 or

3.0); a brief description of that function; the library in which the function is contained; its negative offset from the library base (useful to the "take charge" machine language programmer); its syntax; its C and machine language usage; a description of its arguments (sometimes referred to as parameters); and a description of the value (if any) that is returned by the function. For example, here's the entry for Intuition's OpenWindow( ) function:

OpenWindow

Description:	opens an Intuition window
Library:	intuition.library
Offset:	-$CC
Syntax:	window = OpenWindow(newWindow)
C:	struct Window *OpenWindow(struct NewWindow *)
ML:	d0 = OpenWindow(a0)
Arguments:	newWindow = NewWindow structure that describes window to open
Result:	window = a new Window structure; zero if unsuccessful

The **Syntax** section gives you a general idea of what the function expects and what it returns. If you look at the C description, you can see that OpenWindow( ) expects a pointer to a NewWindow structure and returns a pointer to a Window structure. The **Machine Language** description shows that the Window pointer should be placed in a0 and the pointer to the opened Window is returned in d0 (all functions that return values return them in d0). The following C and machine language programs are examples of how you can use this information to open a window. Five library functions are used in these program: OpenLibrary( ), OpenWindow( ), Wait( ), CloseWindow( ), and CloseLibrary( ).

Program 1-1. C Window

```
/*
   C code that opens a window and waits for you to close it
*/

#include <intuition/intuition.h> /* Include Intuition stuff */

struct IntuitionBase *IntuitionBase = NULL; /* Define IntuitionBase */
struct Window *MyWindow = NULL; /* Define Window pointer */
struct NewWindow MyNewWindow = { /* Set up NewWindow structure */
   0,12,
```

```
    200,100,
    -1,-1,
    CLOSEWINDOW,
    WINDOWCLOSE | WINDOWDEPTH | WINDOWDRAG | WINDOWSIZING | ACTIVATE,
    NULL,
    NULL,
    "Close Me",
    NULL,
    NULL,
    80,24,
    -1,-1,
    WBENCHSCREEN
};

void main( )
{
    /* Open intuition.library */
    if ((IntuitionBase=(struct IntuitionBase *)
        OpenLibrary("intuition.library",0))==NULL)
            exit(0);

    /* Open window */
    if ((MyWindow=(struct Window *)OpenWindow(&MyNewWindow))==NULL) {
        CloseLibrary(IntuitionBase);
        exit(0);
    }

    /* Wait for user to click close box */
    Wait(1<<MyWindow->UserPort->mp_SigBit);

    CloseWindow(MyWindow); /* Close window */
    CloseLibrary(IntuitionBase); /* Close intuition.library */
}
```

Program 1-2. Machine Language Window

```
*Open a window and wait for user to close it

*Include Intuition stuff
    INCLUDE "intuition/intuition.i"

*Address that holds pointer to ExecBase (this is ALWAYS 4)
ExecBase     equ 4

* Hard-coded function offsets—no need to link with amiga.lib with these!
```

```
OpenLibrary  equ -552
CloseLibrary equ -414
OpenWindow   equ -204
CloseWindow  equ -72
Wait         equ -318

   SECTION code,CODE

*Open intuition.library
   movea.l #IntuitionName,a1     ;Library name in a1
   move.l  #0,d0                 ;Version number in d0
   movea.l ExecBase,a6           ;Pointer to ExecBase in a6
   jsr     OpenLibrary(a6)       ;Call OpenLibrary( )
   move.l  d0,_IntuitionBase     ;Save intuition.library base
   beq.s   Abort2                ;Exit program if OpenLibrary fails

*Open a window
   movea.l #MyNewWindow,a0       ;Pointer to NewWindow in a0
   move.l  _IntuitionBase,a6     ;Pointer to IntuitionBase in a6
   jsr     OpenWindow(a6)        ;Call OpenWindow( )
   move.l  d0,MyWindow           ;Save pointer to newly opened window
   beq.s   Abort1                ;Exit program in OpenWindow fails

*Wait for user to click close box
   movea.l MyWindow,a0           ;Pointer to window's
   movea.l wd_UserPort(a0),a0    ;   user port in a0
   move.b  MP_SIGBIT(a0),d1      ;Window's signal bits
   moveq.l #1,d0                 ;Covert bit number to mask
   lsl.l   d1,d0                 ;   and place the result in d0
   movea.l ExecBase,a6           ;Pointer to ExecBase in a6
   jsr     Wait(a6)              ;Call Wait( )

*Close the window
   movea.l MyWindow,a0           ;Pointer to window in a0
   move.l  _IntuitionBase,a6     ;Pointer to IntuitionBase in a6
   jsr     CloseWindow(a6)       ;Call CloseWindow( )

*Close intuition.library
Abort1:
   move.l  _IntuitionBase,a1     ;Pointer to IntuitionBase in a1
   movea.l ExecBase,a6           ;Pointer to ExecBase in a6
   jsr     CloseLibrary(a6)      ;Call CloseLibrary( )

*Exit the program
Abort2:
   clr.l   d0                    ;Set return code
   rts                           ;Drop out of program
```

```
    SECTION data,DATA

*NewWindow structure
MyNewWindow:
    dc.w 0,12
    dc.w 200,100
    dc.b -1,-1
    dc.l CLOSEWINDOW
    dc.l WINDOWCLOSE!WINDOWDEPTH!WINDOWDRAG!WINDOWSIZING!ACTIVATE
    dc.l 0
    dc.l 0
    dc.l WindowTitle
    dc.l 0
    dc.l 0
    dc.w 80,24
    dc.w -1,-1
    dc.w WBENCHSCREEN

IntuitionName:
    dc.b 'intuition.library',0    ;Library name
WindowTitle:
    dc.b 'Close Me',0             ;Text to appear in window's drag bar


    SECTION mem,BSS

_IntuitionBase:
    ds.l 1                        ;Place to store IntuitionBase
MyWindow
    ds.l 1                        ;Place to store Window pointer

    END
```

Machine Language and Function Offsets

In machine language, library functions are called with a JSR using address indirect with displacement addressing mode, which generates the destination address from the contents of register a6. For example, our previous machine language window program contains the following instructions:

```
OpenWindow equ -552
.
move.l _IntuitionBase,a6        ;Pointer to IntuitionBase in a6
jsr    OpenWindow(a6)           ;Call OpenWindow
```

The MOVE instruction puts the base address of the Intuition library in register a6. The label OpenWindow contains the offset of OpenWindow( ) function from the base

address of Intuition. The JSR instruction combines the base address with the offset to produce an actual destination address. By putting the base address in a6, we also tell OpenWindow( ) where its current library base is.

To tell you the truth, our machine language program is slightly unorthodox because it hard codes the library offsets in the beginning of the program using EQU directives. Purists obtain their function offsets using XREF statements. If we were to program this "by the book," the above lines would read

```
XREF _LVOOpenLibrary
.
move.l _IntuitionBase,a6          ;Pointer to IntuitionBase in a6
jsr  _LVOOpenWindow(a6)           ;Call OpenWindow( )
```

so the OpenWindow( ) offset would be obtained at link time. All of the program's library calls could (and some would say, should) be coded this way.

Where does the _LVOOpenLibrary offset come from? Besides containing linked library functions, amiga.lib also contains the function offsets for every Amiga library. That's why so many machine language programs must be linked with amiga.lib. If you use the XREF method, you must precede all function names with an _LVO in order for amiga.lib to recognize them. If you hard-code your offsets as we did in Program 1-2, however, you can call the functions whatever you like—you won't even have to take the time to link with amiga.lib (unless, of course, you used one of amiga.lib's linked library functions).

Final Notes

A few last minute points should be made regarding the terminology used in the following list of functions: By convention, a TRUE value represents a nonzero value (usually equal to 1) while a FALSE or a NULL value is equivalent to 0. For the sake of clarity, this book usually says *zero* instead of NULL or FALSE and *nonzero* instead of TRUE.

Keep in mind that the information in this chapter is mainly for reference purposes—it's a place you can look up information about any Amiga library function, quickly and easily. For the finer details regarding a specific library function, we suggest you refer to Commodore's official Amiga ROM Kernal reference manuals.

AbortIO

Description: attempts to abort an IO request
Library: exec.library
Offset: -$1E0
Syntax: AbortIO(iORequest)
C: VOID AbortIO(struct IORequest *)
ML: AbortIO(a1)
Arguments: iORequest = IORequest to abort
Result: none

AbortPkt (Revision 2.0)

Description: aborts an asynchronous packet
Library: dos.library
Offset: -$108
Syntax: AbortPkt(port, pkt)
C: VOID AbortPkt(struct MsgPort *, struct DosPacket *)
ML: AbortPkt(d1, d2)
Arguments: port = port to which the packet was sent
pkt = the packet you wish aborted
Result: none

ActivateCxObj (Revision 2.0)

Description: changes a commodity object's activation state
Library: commodities.library
Offset: -$2A
Syntax: previous = ActivateCxObj(cxobj, flag)
C: LONG ActivateCxObj(CxObj *, LONG)
ML: d0 = ActivateCxObj(a0, d0)
Arguments: cxobj = commodity object to modify
flag = zero to deactivate object; nonzero to activate
Result: previous = zero if previously active; nonzero otherwise

ActivateGadget

Description: activates a string or custom gadget
Library: intuition.library
Offset: -$1CE
Syntax: success = ActivateGadget(gadget, window, requester)
C: BOOL ActivateGadget(struct Gadget *, struct Window *, struct Requester *)
ML: d0 = ActivateGadget(a0, a1, a2)
Arguments: gadget = gadget to activate
window = window containing the gadget
requester = requester containing gadget; may be 0
Result: success = zero if unsuccessful

ActivateWindow

Description: activates a window
Library: intuition.library
Offset: -$1C2
Syntax: ActivateWindow(window)
C: VOID ActivateWindow(struct Window *)
ML: d0 = ActivateWindow(a0)
Arguments: window = window to activate
Result: none

AddAmigaGuideHostA (Revision 3.0)

Description: adds a host
Library: amigaguide.library
Offset: -$8A
Syntax: handle = AddAmigaGuideHostA(hook, name, tags)
C: AMIGAGUIDEHOST AddAmigaGuideHostA(struct Hook *, STRPTR, struct TagItem *)
ML: d0 = AddAmigaGuideHostA(a0, d0, a1)
Arguments: hook = callback function
name = name of database to add
tags = tag list of attributes—none are currently defined
Result: handle = handle to host allocated; zero if unsucessful

AddAnimOb

Description: adds an AnimOb to the linked list of AnimObs
Library: graphics.library
Offset: -$9C
Syntax: AddAnimOb(anOb, anKey, rastPort)
C: VOID AddAnimOb(struct AnimOb *, struct AnimOb **, struct RastPort *)
ML: AddAnimOb(a0, a1, a2)
Arguments: anOb = AnimOb structure to be added to the list
anKey = address of a pointer to the first AnimOb in the list; 0 if there are no AnimObs in the list so far
rastPort = RastPort structure
Result: none

AddAppIconA (Revision 2.0)

Description: adds an icon to WorkBench's list of AppIcons
Library: workbench.library
Offset: -$3C
Syntax: AppIcon = AddAppIconA(id, userdata, text, msgport, lock, diskobj, tags)
C: struct AppIcon *AddAppIcon(ULONG, ULONG, char *, struct MsgPort *, struct FileLock *, struct DiskObject *, struct TagItem *)
ML: d0 = AddAppIconA(d0, d0, d1, a0, a1, a2, a3, a4)

Arguments: id = user-defined id value
userdata = user-defined data
text = name of the icon
lock = currently unused; should be 0
msgport = a message port to receive AppMessage messages
diskobj = an initialized DiskObject structure
tags = tag list specifying additional attributes:
currently unused and must be 0
Result: AppIcon = an AppIcon structure; zero if unsuccessful

AddAppMenuItemA (Revision 2.0)

Description: adds a menuitem to WorkBench's list of AppMenuItems
Library: workbench.library
Offset: -$48
Syntax: AppMenuItem = AddAppMenuItemA(id, userdata, text, msgport, tags)
C: struct AppMenuItem *AddAppMenuItem(ULONG, ULONG, char *, struct MsgPort *, struct TagItem *)
ML: d0 = AddAppMenuItemA(d0, d1, a0, a1, a2)
Arguments: id = user-defined id value
userdata = user-defined data
text = menu item text
msgport = a message port to receive AppMessage messages
tags = tag list specifying additional attributes:
currently unused and must be 0
Result: AppMenuItem = an AppMenuItem structure; zero if unsuccessful

AddAppWindowA (Revision 2.0)

Description: adds a window to WorkBench's list of AppWindows
Library: workbench.library
Offset: -$30
Syntax: AppWindow = AddAppWindowA(id, userdata, window, msgport, tags)
C: struct AppWindow *AddAppWindow(ULONG, ULONG, struct Window *, struct MsgPort *, struct TagItem *)
ML: d0 = AddAppWindowA(d0, d1, a0, a1, a2)
Arguments: id = user-defined id value
userdata = user-defined data
window = window to add
msgport = a message port to receive AppMessage messages
tags = tag list specifying additional attributes:
currently unused and must be 0
Result: AppWindow = an AppWindow structure; zero if unsuccessful

AddBob

Description: adds a Bob to current gel list
Library: graphics.library
Offset: -$60
Syntax: AddBob(bob, rastPort)
C: VOID AddBob(struct Bob *, struct RastPort *)
ML: AddBob(a0, a1)
Arguments: bob = Bob structure to be added to the gel list
rastPort = RastPort structure
Result: none

AddBootNode (Revision 2.0)

Description: adds a BOOTNODE to the system
Library: expansion.library
Offset: -$24
Syntax: success = AddBootNode(bootPri, flags, deviceNode, configDev)
C: BOOL AddBootNode(BYTE, ULONG, struct DeviceNode *, struct ConfigDev *)
ML: d0 = AddBootNode(d0, d1, a0, a1)
Arguments: bootPri = boot priority for this disk
flags = specifies additional information—ADNF_STARTPROC ($00000001)
deviceNode = initialized DeviceNode structure returned from MakeDosNode( )
configDev = initialized ConfigDev strucutre for the board
Result: success = zero if unsuccessful

AddBuffers (Revision 2.0)

Description: changes the number of filesystem buffers
Library: dos.library
Offset: -$2DC
Syntax: success = AddBuffers(filesystem, number)
C: BOOL AddBuffers(STRPTR, LONG)
ML: d0 = AddBuffers(d1, d2)
Arguments: filesystem = name of device to which to add buffers (with :)
number = number of buffers to add (or subtract, if negative)
Result: success = zero if unsuccessful

AddClass (Revision 2.0)

Description: makes a public boopsi class available to other tasks
Library: intuition.library
Offset: -$2AC
Syntax: AddClass(class)
C: VOID AddClass(struct IClass *)
ML: AddClass(a0)

Arguments: class = pointer returned by MakeClass( )
Result: none

AddConfigDev

Description: adds a ConfigDev structure to the system
Library: expansion.library
Offset: -$1E
Syntax: AddConfigDev(configDev)
C: VOID AddConfigDev(struct ConfigDev *)
ML: AddConfigDev(a0)
Arguments: configDev = ConfigDev structure
Result: none

AddDevice

Description: adds a device to the system
Library: exec.library
Offset: -$1B0
Syntax: AddDevice(device)
C: VOID AddDevice(struct Device *)
ML: AddDevice(a1)
Arguments: device = initialized device node
Result: none

AddDosEntry (Revision 2.0)

Description: adds a DosList entry to AmigaDOS's device list
Library: dos.library
Offset: -$2A6
Syntax: success = AddDosEntry(dlist)
C: LONG AddDosEntry(struct DosList *)
ML: d0 = AddDosEntry(d1)
Arguments: dlist = device list entry to be added
Result: success = zero if unsuccessful

AddDosNode (Obsolete under Revision 2.0)

Description: mounts a disk to the system
Library: expansion.library
Offset: -$96
Syntax: sucess = AddDosNode(bootPri, flags, deviceNode)
C: BOOL AddDosNode(BYTE, ULONG, struct DeviceNode *)
ML: d0 = AddDosNode(d0, d1, a0)
Arguments: bootPri = boot priority for this disk
flags = specifies additional information—ADNF_STARTPROC ($00000001)
deviceNode = initialized DeviceNode structure returned from MakeDosNode( )
Result: success = zero if unsuccessful

AddDTObject (Revision 3.0)

Description: adds a DataType object to a window or requester
Library: datatypes.library
Offset: -$48
Syntax: realposition = AddDTObject(window, requester, object, position)
C: LONG AddDTObject(struct Window *, struct Requester *, Object *, LONG)
ML: d0 = AddDTObject(a0, a1, a2, d0)
Arguments: window = window to receive object
requester—must be zero
object = Object structure returned by NewDTObjectA( )
position = integer position in window's gadget list; -1 to put the object at end
Result: realposition = actual position of gadget

AddFont

Description: adds a font to the system list
Library: graphics.library
Offset: -$1E0
Syntax: AddFont(textFont)
C: VOID AddFont(struct TextFont *)
ML: AddFont(a1)
Arguments: textFont = initialized TextFont structure
Result: none

AddFreeList

Description: adds memory to a free list
Library: icon.library
Offset: -$48
Syntax: success = AddFreeList(free, mem, len)
C: BOOL AddFreeList(struct FreeList *, APTR, ULONG)
ML: d0 = AddFreeList(a0, a1, a2)
Arguments: free = a FreeList structure
mem = the memory to be added to the list
len = number of bytes to add
Result: success = zero if unsuccessful

AddGadget

Description: adds a gadget to a window
Library: intuition.library
Offset: -$2A
Syntax: realPosition = AddGadget(window, gadget, position)
C: UWORD AddGadget(struct Window *, struct Gadget *, UWORD)
ML: d0 = AddGadget(a0, a1, d0)

Arguments: window = window to receive the gadget
gadget = gadget to add
position = position in gadget list where gadget should be inserted;
0 for start of list, -1 for end of list
Result: realPosition = position where gadget is actually added

AddGList

Description: adds a linked list of gadgets to a window or requester
Library: intuition.library
Offset: -$1B6
Syntax: realPosition = AddGList(window, gadget, position, numgad,
requester)
C: UWORD AddGList(struct Window *, struct Gadget *, UWORD, WORD,
struct Requester *)
ML: d0 = AddGList(a0, a1, d0, d1, a2)
Arguments: window = window to receive gadget list
gadget = first gadget in gadget list
position = position in the window's gadget list where the new
gadget should be inserted; 0 for start of list, -1 for end of list
numgad = number of gadgets being added; set to 1 to add the
entire NULL-terminated list
requester = requester to recieve gadgets; may be 0
Result: realPosition = position where gadget list is actually added

AddHead

Description: inserts node at the head of a list
Library: exec.library
Offset: -$F0
Syntax: AddHead(list, node)
C: VOID AddHead(struct List *, struct Node *)
ML: AddHead(a0, a1)
Arguments: list = target list header
node = node to insert at head
Result: none

AddIEvents (Revision 2.0)

Description: adds input events to commodities' input stream
Library: commodities.library
Offset: -$B4
Syntax: AddIEvents(events)
C: VOID AddIEvents(struct InputEvent *)
ML: AddIEvents(a0)
Arguments: events = zero-terminated linked list of input events
Result: none

AddIntServer

Description: adds an interrupt server to the system server chain
Library: exec.library
Offset: -$A8
Syntax: AddIntServer(intNum, interrupt)
C: VOID AddIntServer(ULONG, struct Interrupt *)
ML: AddIntServer(d0, a1)
Arguments: intNum = Paula interrupt bit number (0-14)
interrupt = initialized Interrupt structure
Result: none

AddLibrary

Description: adds a library to the system
Library: exec.library
Offset: -$18C
Syntax: AddLibrary(library)
C: VOID AddLibrary(struct Library *)
ML: AddLibrary(a1)
Arguments: library = initialized Library structure
Result: none

AddMemHandler (Revision 3.0)

Description: adds a low memory handler to Exec
Library: exec.library
Offset: -$306
Syntax: AddMemHandler(memHandler)
C: VOID AddMemHandler(struct Interrupt *)
ML: AddMemHandler(a1)
Arguments: memHandler = initialized Interrupt structure
Result: none

AddMemList

Description: adds memory to the system free pool
Library: exec.library
Offset: -$26A
Syntax: AddMemList(size, attributes, pri, base, name)
C: VOID AddMemList(ULONG, ULONG, LONG, APTR, STRPTR)
ML: AddMemList(d0, d1, d2, a0, a1)
Arguments: size = size of the memory in bytes
attributes = attributes that the memory will have
pri = priority for the memory
base = pointer to new memory area
name = name to be placed in memory header
Result: none

AddNamedObject (Revision 3.0)

Description: adds a named object to a namespace
Library: utility.library
Offset: -$DE
Syntax: success = AddNamedObject(namespace, object)
C: BOOL AddNamedObject(struct NamedObject *, struct NamedObject *)
ML: d0 = AddNamedObject(a0, a1)
Arguments: namespace = the namespace to which to add the object
object = the object to add
Result: success = zero if unsuccessful

AddPart (Revision 2.0)

Description: appends a filename or directory name to the end of a pathname
Library: dos.library
Offset: -$372
Syntax: success = AddPart(pathname, filename, size)
C: BOOL AddPart(STRPTR, STRPTR, ULONG)
ML: d0 = AddPart(d1, d2, d3)
Arguments: pathname = the destination pathname
filename = the filename or directory name to add
Result: success = zero if unsuccessful

AddPort

Description: adds a public message port to the system
Library: exec.library
Offset: -$162
Syntax: AddPort(port)
C: VOID AddPort(struct MsgPort *)
ML: AddPort(a1)
Arguments: port = message port to add
Result: none

AddResource

Description: adds a resource to the system
Library: exec.library
Offset: -$1E6
Syntax: AddResource(resource)
C: VOID AddResource(APTR)
ML: AddResource(a1)
Arguments: resource = resource to add
Result: none

AddSegment (Revision 2.0)

Description: adds a resident segment to the resident list
Library: dos.library

Offset: -$306
Syntax: success = AddSegment(name, seglist, type)
C: BOOL AddSegment(STRPTR, BPTR, LONG)
ML: d0 = AddSegment(d1, d2, d3)
Arguments: name = segment name
seglist = code seglist
type = initial usecount; normally 0
Result: success = zero if unsuccessful

AddSemaphore

Description: initializes and adds a signal semaphore to the system
Library: exec.library
Offset: -$258
Syntax: AddSemaphore(signalSemaphore)
C: VOID AddSemaphore(struct SignalSemaphore *)
ML: AddSemaphore(a1)
Arguments: signalSemaphore = signal semaphore to add
Result: none

AddTail

Description: appends node to tail of a list
Library: exec.library
Offset: -$F6
Syntax: AddTail(list, node)
C: VOID AddTail(struct List *, struct Node *)
ML: AddTail(a0, a1)
Arguments: list = target list header
node = node to insert at tail of the list
Result: none

AddTask

Description: adds a task to the system
Library: exec.library
Offset: -$11A
Syntax: AddTask(task, initialPC, finalPC)
C: APTR AddTask(struct Task *, APTR, APTR)
ML: AddTask(a1, a2, a3)
Arguments: task = task control block (TCB)
initialPC = task's entry point
finalPC = task's exit code; zero to use system's general finalizer
Result: task = address of new task; zero if unsuccessful (valid under Revision 2.0 only)

AddVSprite

Description: adds a VSprite to the current gel list
Library: graphics.library
Offset: -$66
Syntax: AddVSprite(vs, rastPort)
C: VOID AddVSprite(struct VSprite *, struct RastPort *)
ML: AddVSprite(a0, a1)
Arguments: vs = VSprite structure to be added to the gel list
rastPort = RastPort structure
Result: none

Alert

Description: alerts the user of an error
Library: exec.library
Offset: -$6C
Syntax: Alert(alertNum)
C: VOID Alert(ULONG)
ML: Alert(d7)
Arguments: alertNum = number indicating the desired alert
Result: none

AllocAbs

Description: allocates memory from a specific location
Library: exec.library
Offset: -$CC
Syntax: memoryBlock = AllocAbs(byteSize, location)
C: VOID *AllocAbs(ULONG, APTR)
ML: d0 = AllocAbs(d0, a1)
Arguments: byteSize = number of bytes to allocate
location = address of desired memory
Result: memoryBlock = allocated memory block; zero if unsuccessful

AllocAslRequest (Revision 2.0)

Description: allocates an ASL requester
Library: asl.library
Offset: -$30
Syntax: requester = AllocAslRequest(reqType, tags)
C: APTR AllocAslRequest(ULONG, struct TagItem *)
ML: d0 = AllocAslRequest(d0, a0)
Arguments: reqType = type of requester to allocate:
ASL_FileRequest ($00000000),
ASL_FontRequest ($00000001),

ASL_ScreenModeRequest (Revision 3.0) ($00000002)
tags = tag list specifying attributes:
see <libraries/asl.h> for description
Result: requester = pointer to initialized requester structure; zero if unsuccessful

Allocate

Description: allocates a block of memory
Library: exec.library
Offset: -$BA
Syntax: memoryBlock = Allocate(memHeader, byteSize)
C: VOID *Allocate(struct MemHeader *, ULONG)
ML: d0 = Allocate(a0, d0)
Arguments: memHeader = local memory list header
byteSize = size of desired block in bytes
Result: memoryBlock = allocated memory block; zero if unsuccessful

AllocateTagItems (Revision 2.0)

Description: allocates a tag list
Library: utility.library
Offset: -$42
Syntax: tagList = AllocateTagItems(numTags)
C: struct TagItem *AllocateTagItems(ULONG)
ML: d0 = AllocateTagItems(d0)
Arguments: numTags = the number of TagItem structures to allocate
Result: tagList = list of new TagItem structures; zero if unsuccessful

AllocBitMap (Revision 3.0)

Description: allocates a bitmap and accompanying bitplanes
Library: graphics.library
Offset: -$396
Syntax: bitmap = AllocBitMap(width, height, depth, flags, friendBitmap)
C: struct BitMap *AllocBitMap(ULONG, ULONG, ULONG, ULONG, struct BitMap *)
ML: d0 = AllocBitMap(d0, d1, d2, d3, a0)
Arguments: width, height = bitmap dimensions
depth = number of bitplanes; note that you may get more than you ask for!
flags = bipmap options—BMF_CLEAR ($00000001), BMF_DISPLAYABLE ($00000002), BMF_INTERLEAVED ($00000004), BMF_STANDARD ($00000008), BMF_MINPLANES ($00000010)
friendBitmap = a bitmap that holds graphics that may be blitted to the bitmap being allocated
Result: none

AllocConfigDev

Description: allocates a ConfigDev structure
Library: expansion.library
Offset: -$30
Syntax: configDev = AllocConfigDev( )
C: struct ConfigDev *AllocConfigDev( )
ML: d0 = AllocConfigDev( )
Arguments: none
Result: configDev = cleared ConfigDev structure; zero if unsuccessful

AllocDBufInfo (Revision 3.0)

Description: allocates a DBufInfo structure in preparation for double buffering
Library: graphics.library
Offset: -$3C6
Syntax: db = AllocDBufInfo(viewPort)
C: struct DBufInfo *AllocDBufInfo(struct ViewPort *)
ML: d0 = AllocDBufInfo(a0)
Arguments: viewPort = ViewPort structure
Result: none

AllocDosObject (Revision 2.0)

Description: creates a DOS object
Library: DOS.library
Offset: -$E4
Syntax: ptr = AllocDosObject(type, tags)
C: VOID *AllocDosObjectTags(ULONG, Struct TagItem*)
ML: d0 = AllocDosObject(d1, d2)
Arguments: type = type of object
see <dos/dos.h> for description
tags = tag list specifying additional attributes:
see <dos/dostags.h> for description
Result: packet = the new object; zero if unsuccessful

AllocEntry

Description: allocates multiple regions of memory
Library: exec.library
Offset: -$DE
Syntax: memList = AllocEntry(memList)
C: struct MemList *AllocEntry(struct MemList *)
ML: d0 = AllocEntry(a0)
Arguments: memList = MemList structure filled in with MemEntry structures
Result: memList = MemList structure filled in with allocated memory

AllocExpansionMem

Description: allocates expansion memory
Library: expansion.library
Offset: -$36
Syntax: startSlot = AllocExpansionMem(numSlots, slotOffset)
C: APTR AllocExpansionMem(ULONG, ULONG)
ML: d0 = AllocExpansionMem(d0, d1)
Arguments: numSlots = number of slots requested
slotOffset = an offset for startSlot
Result: startSlot = the slot number allocated; -1 if unsuccessful

AllocFileRequest (Revision 2.0—Obsolete under Revision 3.0)

Description: allocates a FileRequester structure
Library: asl.library
Offset: -$1E
Syntax: requester = AllocFileRequest( )
C: struct FileRequester *AllocFileRequest(VOID)
ML: d0 = AllocFileRequest( )
Arguments: none
Result: requester = pointer to initialized FileRequester structure; zero if unsuccessful

AllocIFF (Revision 2.0)

Description: allocates a new IFFHandle structure
Library: iffparse.library
Offset: -$1E
Syntax: iff = AllocIFF( )
C: struct IFFHandle *AllocIFF(VOID)
ML: d0 = AllocIFF( )
Arguments: none
Result: iff = a new IFFHandle structure; zero if unsuccessful

AllocLocalItem (Revision 2.0)

Description: allocates a local context item structure
Library: iffparse.library
Offset: -$BA
Syntax: item = AllocLocalItem(type, id, ident, dataSize)
C: struct LocalContextItem *AllocLocalItem(LONG, LONG, LONG, LONG)
ML: d0 = AllocLocalItem(d0, d1, d2, d3)
Arguments: type = additional identification value
id = additional identification value
ident = identifier for class of context item
dataSize = number of bytes of user data to allocate
Result: item = an initialized LocalContextItem; zero if unsuccessful

AllocMem

Description: allocates memory
Library: exec.library
Offset: -$C6
Syntax: memoryBlock = AllocMem(byteSize, attributes)
C: VOID *AllocMem(ULONG, ULONG)
ML: d0 = AllocMem(d0, d1)
Arguments: byteSize = number of bytes required
attributes = type of memory—MEMF_ANY ($00000000), MEMF_PUBLIC ($00000001), MEMF_CHIP ($00000002), MEMF_FAST ($00000004), MEMF_LOCAL ($00000008), MEMF_24BITDMA ($00000010), MEMF_CLEAR ($00010000)
Result: memoryBlock = allocated memory block

AllocNamedObjectA (Revision 3.0)

Description: allocates a utility library object
Library: utility.library
Offset: -$E4
Syntax: object = AllocNamedObjectA(name, tags)
C: struct NamedObject *AllocNamedObject(STRPTR, struct TagItem *)
ML: d0 = AllocNamedObjectA(a0, a1)
Arguments: name = name for the object; must not be 0
tags = tag list specifying attributes—ANO_NameSpace ($00004000), ANO_UserSpace ($00004001), ANO_Priority ($00004002), ANO_Flags ($00004003); defined flags: NSF_NODUPS ($00000001), NSF_CASE ($00000002)
Result: object = the new NamedObject structure; zero if unsuccessful

AllocPooled (Revision 3.0)

Description: allocates memory with the pool manager
Library: exec.library
Offset: -$2C4
Syntax: memoryBlock = AllocPooled(poolHeader, memSize)
C: VOID *AllocPooled(VOID *,ULONG)
ML: d0 = AllocPooled(a0, d0)
Arguments: memSize = the number of bytes to allocate
poolHeader = a specific private pool header
Result: memoryBlock = allocated memory block; longword-aligned

AllocRaster

Description: allocates space for a bitplane
Library: graphics.library
Offset: -$1EC
Syntax: planeptr = AllocRaster(width, height)
C: PLANEPTR AllocRaster(UWORD, UWORD)
ML: d0 = AllocRaster(d0, d1)

Arguments: width, height = dimensions of bitplane
Result: planeptr = pointer to allocated bitplane; zero if unsuccessful

AllocRemember

Description: allocates memory, recording the allocation
Library: intuition.library
Offset: -$18C
Syntax: memBlock = AllocRemember(rememberKey, size, flags)
C: APTR AllocRemember(struct Remember **, ULONG, ULONG)
ML: d0 = AllocRemember(a0, d0, d1)
Arguments: rememberKey = address of a pointer to a Remember structure; should be 0 on first call to this function
size = number of bytes to allocate
flags = bit flags specifying the type of memory to allocate—MEMF_ANY ($00000000), MEMF_PUBLIC ($00000001), MEMF_CHIP ($00000002), MEMF_FAST ($00000004), MEMF_LOCAL ($00000100), MEMF_24BITDMA ($00000200), MEMF_CLEAR ($00010000)
Result: memBlock = the allocated memory; zero if unsuccessful

AllocScreenBuffer (Revision 3.0)

Description: allocates a ScreenBuffer for double buffering
Library: intuition.library
Offset: -$300
Syntax: sBuffer = AllocScreenBuffer(screen, bitMap, flags)
C: struct ScreenBuffer *AllocScreenBuffer(struct Screen *, struct BitMap *, ULONG)
ML: d0 = AllocScreenBuffer(a0, a1, d0)
Arguments: screen = screen to double buffer
bitMap = BitMap structure for CUSTOMBITMAP screens, otherwise NULL
flags = bitmap options—SB_SCREEN_BITMAP ($00000001), SB_COPY_BITMAP ($00000002)
Result: sBuffer = a ScreenBuffer structure; zero if unsuccessful

AllocSignal

Description: allocates a signal bit
Library: exec.library
Offset: -$14A
Syntax: signalNum = AllocSignal(signalNum)
C: BYTE AllocSignal(BYTE)
ML: d0 = AllocSignal(d0)
Arguments: signalNum = the desired signal number; 0-31, or -1 for no preference
Result: signalNum = the signal bit number allocated; 0-31, or -1 if no signals are available

AllocSpriteDataA (Revision 3.0)

Description: allocates memory for a sprite, using a bitmap to define its shape
Library: graphics.library
Offset: -$3FC
Syntax: spritePtr = AllocSpriteDataA(bitmap, tags)
C: spritePtr = AllocSpriteDataA(struct BitMap *, struct TagItem *)
ML: d0 = AllocSpriteDataA(a2, a1)
Arguments: bitmap = bitmap that defines sprite shape
tags = tag list of attributes—SPRITEA_Width ($81000000), SPRITEA_XReplication ($81000002), SPRITEA_YReplication ($81000004), SPRITEA_OutputHeight ($81000006), SPRITEA_Attached ($81000008), SPRITEA_OldDataFormat ($8100000A)
Result: spritePtr = an ExtSprite structure; zero if unsuccessful

AllocTrap

Description: allocates a processor trap vector
Library: exec.library
Offset: -$156
Syntax: trapNum = AllocTrap(trapNum)
C: LONG AllocTrap(LONG)
ML: d0 = AllocTrap(d0)
Arguments: trapNum = the desired trap number; 0-15, or -1 for no preference
Result: trapNum = the trap number allocated; 0-15, or -1 if no traps are available

AllocVec (Revision 2.0)

Description: allocates memory and keeps track of the size
Library: exec.library
Offset: -$2AC
Syntax: memoryBlock = AllocVec(byteSize, attributes)
C: VOID *AllocVec(ULONG, ULONG)
ML: d0 = AllocVec(d0, d1)
Arguments: byteSize = number of bytes required
attributes = type of memory—MEMF_ANY ($00000000), MEMF_PUBLIC ($00000001), MEMF_CHIP ($00000002), MEMF_FAST ($00000004), MEMF_LOCAL ($00000008), MEMF_24BITDMA ($00000010), MEMF_CLEAR ($00010000)
Result: memoryBlock = allocated memory block

Amiga2Date (Revision 2.0)

Description: converts a system timestamp into a ClockData structure
Library: utility.library
Offset: -$78
Syntax: Amiga2Date(seconds, result)
C: VOID Amiga2Date(ULONG, struct ClockData *)
ML: Amiga2Date(d0, a0)

Arguments: seconds = the timestamp to convert
result = the ClockData structure to initialize
Result: none

AmigaGuideSignal (Revision 3.0)
Description: obtains a signal bit for messages
Library: amigaguide.library
Offset: -$48
Syntax: signal = AmigaGuideSignal(handle)
C: ULONG AmigaGuideSignal(AMIGAGUIDECONTEXT)
ML: d0 = AmigaGuideSignal(a0)
Arguments: handle = handle to an AmigaGuide system
Result: signal = signal bit assigned to database

AndRectRegion
Description: ANDs a rectangle with a region, leaving the result in the region
Library: graphics.library
Offset: -$1F8
Syntax: AndRectRegion(region, rectangle)
C: VOID AndRectRegion(struct Region *, struct Rectangle *)
ML: AndRectRegion(a0, a1)
Arguments: region = destination Region structure
rectangle = Rectangle structure
Result: none

AndRegionRegion
Description: ANDs two regions together, leaving the result in the second region
Library: graphics.library
Offset: -$270
Syntax: success = AndRegionRegion(region1, region2)
C: BOOL AndRegionRegion(struct Region *, struct Region *)
ML: d0 = AndRegionRegion(a0, a1)
Arguments: region1 = Region structure
region2 = destination Region structure
Result: success = zero if unsuccessful

Animate
Description: processes every AnimOb in the current animation list
Library: graphics.library
Offset: -$A2
Syntax: Animate(anKey, rastPort)
C: VOID Animate(struct AnimOb **, struct RastPort *)
ML: Animate(a0, a1)

Arguments: ankey = address of a pointer to the head AnimOb
rastPort = RastPort structure
Result: none

ApplyTagChanges (Revision 3.0)
Description: modifies a tag list based on a second tag list
Library: utility.library
Offset: -$BA
Syntax: ApplyTagChanges(list, changeList)
C: VOID ApplyTagChanges(struct TagItem *, struct TagItem *)
ML: ApplyTagChanges(a0, a1)
Arguments: list = the tag list to modify
changeList = the tag list used to modify list
Result: none

AreaDraw
Description: adds a point to a RastPort's AreaInfo structure
Library: graphics.library
Offset: -$102
Syntax: error = AreaDraw(rastPort, x, y)
C: ULONG AreaDraw(struct RastPort *, WORD, WORD)
ML: d0 = AreaDraw(a1, d0, d1)
Arguments: rastPort = RastPort structure
x, y = point coordinates
Result: error = zero if successful

AreaEllipse
Description: adds an ellipse to a RastPort's AreaInfo structure
Library: graphics.library
Offset: -$BA
Syntax: error = AreaEllipse(rastPort, cx, cy, width, height)
C: LONG AreaEllipse(struct RastPort *, WORD, WORD, WORD, WORD)
ML: d0 = AreaEllipse(a1, d0, d1, d2, d3)
Arguments: rastPort = RastPort structure
cx, cy = ellipse's centerpoint relative to the rastport
width = horizontal radius of the ellipse; must be greater than zero
height = vertical radius of the ellipse; must be greater than zero
Result: error = zero if successful

AreaEnd
Description: fills the polygons and ellipses described by a RastPort's AreaInfo structure
Library: graphics.library
Offset: -$108
Syntax: error = AreaEnd(rastPort)
C: LONG AreaEnd(struct RastPort *)

ML: d0 = AreaEnd(a1)
Arguments: rastPort = RastPort structure with AreaInfo
Result: error = zero if successful

AreaMove
Description: defines the first point of a polygon in a RastPort's AreaInfo structure
Library: graphics.library
Offset: -$FC
Syntax: error = AreaMove(rastPort, x, y)
C: LONG AreaMove(struct RastPort *, WORD, WORD)
ML: d0 = AreaMove(a1, d0, d1)
Arguments: rastPort = a RastPort structure
x, y = point coordinates in the raster
Result: error = zero if successful

AskFont
Description: returns a RastPort's current font attributes
Library: graphics.library
Offset: -$1DA
Syntax: AskFont(rastPort, textAttr)
C: VOID AskFont(struct RastPort *, struct TextAttr *)
ML: AskFont(a1, a0)
Arguments: rastPort = RastPort in question
textAttr = TextAttr structure to receive font data
Result: none

AskKeyMapDefault (Revision 2.0)
Description: retrieves a pointer to the current default key map
Library: keymap.library
Offset: -$24
Syntax: keyMap = AskKeyMapDefault( )
C: struct KeyMap *AskKeyMapDefault(VOID)
ML: d0 = AskKeyMapDefault( )
Arguments: none
Result: keyMap = a permanently allocated KeyMap structure

AskSoftStyle
Description: returns a RastPort's soft style bits for the current font
Library: graphics.library
Offset: -$54
Syntax: enable = AskSoftStyle(rastPort)
C: ULONG AskSoftStyle(struct RastPort *)
ML: d0 = AskSoftStyle(a1)
Arguments: rastPort = RastPort in question
Result: enable = type bits

AslRequest (Revision 2.0)

Description: uses an ASL requester to get user input
Library: asl.library
Offset: -$3C
Syntax: result = AslRequest(requester, tags)
C: BOOL AslRequest(APTR, struct TagItem *)
ML: d0 = AslRequest(a0, a1)
Arguments: requester = requester structure returned by AllocAslRequest( )
tags = tag list specifying attributes:
see <libraries/asl.h> for description
Result: result = zero if requester cancelled, nonzero otherwise

AssignAdd (Revision 2.0)

Description: adds a lock to an assign for multidirectory assigns
Library: dos.library
Offset: -$276
Syntax: success = AssignAdd(name, lock)
C: BOOL AssignAdd(STRPTR,BPTR)
ML: d0 = AssignAdd(d1, d2)
Arguments: name = name of device to which to assign the lock (without :)
lock = lock associated with the assigned name
Result: success = zero if unsuccessful

AssignLate (Revision 2.0)

Description: creates an assignment to a path that will be specified later
Library: dos.library
Offset: -$26A
Syntax: success = AssignLate(name, path)
C: BOOL AssignLate(STRPTR, STRPTR)
ML: d0 = AssignLate(d1, d2)
Arguments: name = name of device to be assigned (without :)
path = name of late assignment to be resolved on the first reference
Result: success = zero if unsuccessful

AssignLock (Revision 2.0)

Description: creates an assignment to a locked object
Library: dos.library
Offset: -$264
Syntax: success = AssignLock(name, lock)
C: BOOL AssignLock(STRPTR, BPTR)
ML: d0 = AssignLock(d1, d2)
Arguments: name = name of device to assign lock to (without :)
lock = lock associated with the assigned name
Result: success = zero if unsuccessful

AssignPath (Revision 2.0)

Description: creates an assignment to a specified path
Library: dos.library
Offset: -$270
Syntax: success = AssignPath(name, path)
C: BOOL AssignPath(STRPTR, STRPTR)
ML: d0 = AssignPath(d1, d2)
Arguments: name = name of device to be assigned (without :)
path = name of late assignment to be resolved at each reference
Result: success = zero if unsuccessful

AttachCxObj (Revision 2.0)

Description: attaches a commodity object to an existing list
Library: commodities.library
Offset: -$54
Syntax: AttachCxObj(headObj, cxobj)
C: VOID AttachCxObj(CxObj *, CxObj *)
ML: AttachCxObj(a0, a1)
Arguments: headObj = linked list of objects
cxobj = object to add to the list
Result: none

AttachPalExtra (Revision 3.0)

Description: allocates and attaches a palette sharing structure to a colormap
Library: graphics.library
Offset: -$342
Syntax: error = AttachPalExtra(colorMap, viewPort)
C: LONG AttachPalExtra(Struct ColorMap *, struct ViewPort *)
ML: d0 = AttachPalExtra(a0, a1)
Arguments: colorMap = color map created by GetColorMap( ) function
Result: error = zero if successful

AttemptLockDosList (Revision 2.0)

Description: attempts to lock the DosLists
Library: dos.library
Offset: -$29A
Syntax: dlist = AttemptLockDosList(flags)
C: struct DosList *AttemptLockDosList(ULONG)
ML: d0 = AttemptLockDosList(d1)
Arguments: flags = types of entries you want to lock:
LDF_DEVICES ($00000004), LDF_VOLUMES ($00000008),
LDF_ASSIGNS ($00000010), LDF_ENTRY ($00000020)
Result: dlist = pointer that you pass to NextDosEntry( ) to access DosList;
zero if unsuccessful

AttemptLockLayerRom
Description: attempts to lock a Layer structure
Library: graphics.library
Offset: -$28E
Syntax: success = AttemptLockLayerRom(layer)
C: BOOL AttempLockLayerRom(struct Layer *)
ML: d0 = AttemptLockLayerRom(a5)
Arguments: layer = Layer structure to lock
Result: success = zero if unsuccessful

AttemptRemNamedObject (Revision 3.0)
Description: attempts to remove a named object from a namespace
Library: utility.library
Offset: -$EA
Syntax: result = AttemptRemNamedObject(object)
C: LONG AttemptRemNamedObject(struct NamedObject *)
ML: d0 = AttemptRemNamedObject(a0)
Arguments: object = the object to attempt to remove
Result: success = zero if unsuccessful

AttemptSemaphore
Description: tries to obtain a semaphore without blocking it if unsuccessful
Library: exec.library
Offset: -$240
Syntax: success = AttemptSemaphore(signalSemaphore)
C: LONG AttemptSemaphore(struct SignalSemaphore *)
ML: d0 = AttemptSemaphore(a0)
Arguments: signalSemaphore = an initialized signal semaphore structure
Result: success = zero if unsuccessful

AttemptSemaphoreShared (Revision 2.0)
Description: tries to obtain a shared semaphore without blocking it if unsuccessful
Library: exec.library
Offset: -$2D0
Syntax: success = AttemptSemaphoreShared(signalSemaphore)
C: LONG AttemptSemaphoreShared(struct SignalSemaphore *)
ML: d0 = AttemptSemaphoreShared(a0)
Arguments: signalSemaphore = an initialized signal semaphore structure
Result: success = zero if unsuccessful

AutoRequest
Description: automatically opens a requester and returns user's response
Library: intuition.library
Offset: -$15C

Syntax: response = AutoRequest(window, bodyText, posText, negText, posFlags, negFlags, width, height)
C: BOOL AutoRequest(struct Window *, struct IntuiText *, struct IntuiText *, struct IntuiText *, ULONG, ULONG, WORD, WORD)
ML: d0 = AutoRequest(a0, a1, a2, a3, d0, d1, d2, d3)
Arguments: window = window to own requester
bodyText = IntuiText structure containing main requester text
posText = IntuiText structure containing text for positive-response gadget
negText = IntuiText structure containing text for positive-response gadget
posFlags = IDCMP flags for the positive-response gadget
negFlags = IDCMP flags for the negative-response gadget
width, height = size of requester (ignored under Revision 2.0)
Result: response = nonzero if positive-response gadget is selected; zero if negative-response gadget is selected

AvailFonts

Description: retrieves available disk and memory fonts
Library: diskfont.library
Offset: -$24
Syntax: error = AvailFonts(buffer, bufBytes, flags)
C: LONG AvailFonts(struct AvailFontsHeader *, LONG, ULONG)
ML: d0 = AvailFonts(a0, d0, d1)
Arguments: buffer = memory to be filled with AvailFontsHeader structure and an array of AvailFonts structures
bufBytes = number of bytes in buffer
flags = specifies what kind(s) of fonts to look for:
AFF_MEMORY ($00000001), AFF_DISK ($00000002), AFF_SCALED ($00000004), AFF_BITMAP ($00000008), AFF_TAGGED ($00010000)
Result: error = zero if successful; if unsuccessful, specifies the number of extra bytes needed for buffer

AvailMem

Description: calculates the amount of memory available
Library: exec.library
Offset: -$D8
Syntax: size = AvailMem(attributes)
C: ULONG AvailMem(ULONG)
ML: d0 = AvailMem(d1)
Arguments: requirements = a mask as specified in AllocMem( );
MEMF_LARGEST ($00020000) is an additional flag which, when present in the mask, causes AvailMem( ) to return the size of the largest contiguous memory block matching the requirements
Result: size = total free space remaining (or the largest free block)

BeginRefresh

Description: sets up a window for optimized refreshing
Library: intuition.library
Offset: -$162
Syntax: BeginRefresh(window)
C: VOID BeginRefresh(struct Window *)
ML: BeginRefresh(a0)
Arguments: window = window that needs refreshing
Result: none

BeginUpdate

Description: prepares to repair damaged layer
Library: layers.library
Offset: -$4E
Syntax: success = BeginUpdate(layer)
C: LONG BeginUpdate(struct Layer *)
ML: d0 = BeginUpdate(a0)
Arguments: layer = the layer to update
Result: success = zero if unsuccessful

BehindLayer

Description: moves a layer behind other layers
Library: layers.library
Offset: -$36
Syntax: success = BehindLayer(dummy, layer)
C: LONG BehindLayer(LONG, struct Layer *)
ML: d0 = BehindLayer(a0, a1)
Arguments: dummy = unused
layer = the layer to move
Result: success = zero if unsuccessful

BestModeIDA (Revision 3.0)

Description: calculates the best ModeID for a screen display
Library: graphics.library
Offset: -$41A
Syntax: id = BestModeIDA(tags)
C: ULONG BestModeID(struct TagItem *)
ML: d0 = BestModeIDA(a0)
Arguments: tags = tag list of attributes—BIDTAG_DIPFMustHave ($80000001), BIDTAG_DIPFMustNotHave ($80000002), BIDTAG_ViewPort ($80000003), BIDTAG_NominalWidth ($80000004), BIDTAG_NominalHeight ($80000005), BIDTAG_DesiredWidth ($80000006), BIDTAG_DesiredHeight ($80000007), BIDTAG_Depth ($80000008), BIDTAG_MonitorID ($80000009),

BIDTAG_SourceID ($8000000A), BIDTAG_RedBits ($8000000B), BIDTAG_BlueBits ($8000000C), BIDTAG_GreenBits (EQU $8000000D)
Result: id = ModeID for the best mode to use; -1 if unsuccessful

BitMapScale (Revision 2.0)

Description: resizes a bitmap by removing or expanding pixels
Library: graphics.library
Offset: -$2A6
Syntax: BitMapScale(bitScaleArgs)
C: VOID BitMapScale(struct BitScaleArgs *)
ML: BitMapScale(a0)
Arguments: bitScaleArgs = BitScaleArgs structure containing bitmap and scaling information
Result: none

BltBitMap

Description: uses the blitter to copy (blit) a rectangular area from one BitMap to another
Library: graphics.library
Offset: -$1E
Syntax: planes = BltBitMap(srcBitMap, srcX, srcY, destBitMap, destX, destY, sizeX, sizeY, minterm, mask, tempA)
C: ULONG BltBitMap(struct BitMap *, WORD, WORD, struct BitMap *, WORD, WORD, WORD, WORD, UBYTE, UBYTE, UWORD *)
ML: d0 = BltBitMap(d0, a0, d0, d1, a1, d2, d3, d4, d5, d6, d7, a2)
Arguments: srcBitMap = source bitmap
srcX, srcY = upper left coordinate of source rectangle
destBitMap = destination bitmap
destX, destY = upper left coordinate of destination rectangle
sizeX, sizeY = size of rectangle to blit
minterm = blitter logic function to apply to rectangle
mask = bit mask (color value) defining bitplanes to effect
tempA = pointer to chip memory to buffer one line or NULL
Result: planes = actual number of bitplanes involved in blit

BltBitMapRastPort

Description: uses the blitter to copy (blit) a rectangular area from a BitMap to a RastPort
Library: graphics.library
Offset: -$25E
Syntax: BltBitMapRastPort(srcBitMap, srcX, srcY, destRastPort, destX, destY, sizeX, sizeY, minterm)
C: VOID BltBitMapRastPort(struct BitMap *, WORD, WORD, struct RastPort *, WORD, WORD, WORD, WORD, UBYTE)
ML: d0 = BltBitMapRastPort(d0, a0, d0, d1, a1, d2, d3, d4, d5, d6)

Arguments: srcBitMap = source bitmap
srcX, srcY = upper left coordinate of source rectangle
destRastPort = destination RastPort
destX, destY = upper left coordinate of destination rectangle
sizeX, sizeY = size of rectangle to blit
minterm = blitter logic function to apply to rectangle
Result: none

BltClear

Description: uses blitter to fill a block of memory with zeros (or any other value, under Revision 2.0 and 3.0)
Library: graphics.library
Offset: -$12C
Syntax: BltClear(mem, size, flags)
C: VOID BltClear(VOID *, ULONG, ULONG)
ML: BltClear(a1, d0, d1)
Arguments: mem = memory to be cleared
size = number of bytes to zero out; must be even
flags = set bit 0 to force function to wait until memory is cleared;
set bit 1 if the size argument should be interpreted as the number of rows (upper sixteen bits) and the number of bytes per row (lower sixteen bits)
Result: none

BltMaskBitMapRastPort

Description: uses the blitter to copy (blit) a rectangular area from a BitMap to a RastPort through a mask
Library: graphics.library
Offset: -$27C
Syntax: BltMaskBitMapRastPort(srcBitMap, srcX, srcY, destRastPort, destX, destY, sizeX, sizeY, minterm)
C: VOID BltMaskBitMapRastPort(struct BitMap *, WORD, WORD, struct RastPort *, WORD, WORD, WORD, WORD, UBYTE, APTR)
ML: d0 = BltMaskBitMapRastPort(d0, a0, d0, d1, a1, d2, d3, d4, d5, d6, a2)
Arguments: srcBitMap = source bitmap
srcX, srcY = upper left coordinate of source rectangle
destRastPort = destination RastPort
destX, destY = upper left coordinate of destination rectangle
sizeX, sizeY = size of rectangle to blit
minterm = blitter logic function to apply to rectangle
mask = single bitplane mask
Result: none

BltPattern

Description: draws through a mask using standard drawing rules
Library: graphics.library

Offset: -$138
Syntax: BltPattern(destRastPort, mask, x1, y1, x2, y2, width)
C: VOID BltPattern(struct RastPort *, VOID *, WORD, WORD, WORD, WORD, WORD)
ML: BltPattern(a1, a0, d0, d1, d2, d3, d4)
Arguments: destRastPort = points to the destination RastPort for the blit
mask = single bitplane mask
x1, y1 = upper left corner of rectangular region in RastPort
x2, y2 = lower right corner of rectangular region in RastPort
width = byte width of mask
Result: none

BltTemplate

Description: "cookie cuts" a shape in a rectangle to a RastPort
Library: graphics.library
Offset: -$24
Syntax: BltTemplate(srcTemplate, srcX, srcMod, destRastPort, destX, destY, sizeX, sizeY)
C: VOID BltTemplate(UWORD *, WORD, WORD, struct RastPort *, WORD, WORD, WORD, WORD)
ML: BltTemplate(rastPort, a0, d0, d1, a1, d2, d3, d4, d5)
Arguments: srcTemplate = template mask
srcX = horizontal offset into template mask; 0-15
srcMod = number of bytes per row in template mask
destRastPort = destination RastPort
destX, destY = upper left corner of rectangular region in RastPort
sizeX, sizeY = size of the rectangular template
Result: none

BuildEasyRequestArgs (Revision 2.0)

Description: creates a system requester
Library: intuition.library
Offset: -$252
Syntax: reqWindow = BuildEasyRequestArgs(refWindow, easyStruct, idcmp, args)
C: struct Window *BuildEasyRequest(struct Window *, struct EasyStruct *, ULONG, APTR,)
ML: d0 = BuildEasyRequestArgs(a0, a1, d0, a3)
Arguments: window = window on the screen to receive the requester; 0 to indicate the Workbench (or current Revision 2.0 public screen)
easyStruct = initialized EasyStruct structure
idcmp = IDCMP flags for requester
args = arguments for format commands
Result: reqWindow = a system requester window; zero or 1 if unsuccessful

BuildSysRequest

Description: builds and displays a system requester
Library: intuition.library
Offset: -$168
Syntax: reqWindow = BuildSysRequest(window, bodyText, posText, negText, IDCMPFlags, width, height)
C: struct Window *BuildSysRequest(struct Window *, struct IntuiText *, struct IntuiText *, struct IntuiText *, ULONG, WORD, WORD)
ML: d0 = BuildSysRequest(a0, a1, a2, a3, d0, d1, d2)
Arguments: window = window to contain requester; 0 indicates the Workbench (or current Revision 2.0 public screen)
bodyText = IntuiText structure containing requester's main text
posText = IntuiText structure containing text for positive-response gadget
negText = IntuiText structure containing text for positive-response gadget
idcmpFlags = IDCMP flags for initialization of window containing requester
width, height = size of requester
Result: reqWindow = a window containing the requester
If a requester could not be opened, DisplayAlert( ) is called.
This function then returns 1 if the left mouse button is pressed, 0 if the right mouse button is pressed.

BumpRevision

Description: reformats a name for a copy of a file
Library: icon.library
Offset: -$6C
Syntax: result = BumpRevision(newbuf, oldname)
C: char *BumpRevision(char *, char *)
ML: d0 = BumpRevision(a0, a1)
Arguments: newbuf = buffer to store updated name; should be at least 31 characters
oldname = name to modify
Result: result = pointer to newbuf

CacheClearE (Revision 2.0)

Description: clears the cache with extended control
Library: exec.library
Offset: -$282
Syntax: CacheClearE(address, length, caches)
C: VOID CacheClearE(APTR, ULONG, ULONG)
ML: CacheClearE(a0, d0, d1)
Arguments: address = address at which to begin the operation;
this may be rounded due to hardware granularity
length = length of area to be cleared, or $FFFFFFFF to indicate all addresses should be cleared
caches = bit flags to indicate what caches to affect:
see <exec/execbase.h> for description
Result: none

CacheClearU (Revision 2.0)

Description: clears the cache
Library: exec.library
Offset: -$27C
Syntax: CacheClearU( )
C: VOID CacheClearU(VOID)
ML: CacheClearU( )
Arguments: none
Result: none

CacheControl

Description: controls the instruction and data caches
Library: exec.library
Offset: -$288
Syntax: oldBits = CacheControl(cacheBits, cacheMask)
C: ULONG CacheControl(ULONG,ULONG)
ML: d0 = CacheControl(d0, d1)
Arguments: cacheBits = new values for the bits specified in cacheMask:
see <exec/execbase.h> for description
Result: oldBits = the prior values for all settings

CachePostDMA (Revision 2.0)

Description: updates the cache after hardware DMA
Library: exec.library
Offset: -$300
Syntax: CachePostDMA(vaddress, length, flags)
C: CachePostDMA(APTR, LONG *, ULONG)
ML: CachePostDMA(a0, a1, d0)
Arguments: address = address at which to begin operation
length = pointer to the number of bytes to be affected
flags = bits indicating what action should be taken:
see <exec/execbase.h> for description
Result: none

CachePreDMA (Revision 2.0)

Description: updates the cache prior to hardware DMA
Library: exec.library
Offset: -$2FA
Syntax: paddress = CachePreDMA(vaddress, length, flags)
C: APTR CachePreDMA(APTR,LONG *,ULONG)
ML: d0 = CachePreDMA(a0, a1, d0)
Arguments: vaddress = virtual address at which to begin operation
length = pointer to the number of bytes to be affected
flags = bits indicating what action should be taken:
see <exec/execbase.h> for description

Result: paddress = physical address corresponding to vaddress
length = the contiguous length of physical memory at paddress;
this may be smaller than the requested length—to get the mapping for the next chunk of memory, call the function again with a new address, length, and the DMA_Continue flag

CalcIVG (Revision 3.0)

Description: calculates the number of blank lines above a ViewPort
Library: graphics.library
Offset: -$33C
Syntax: count = CalcIVG(view, viewPort)
C: UWORD CalcIVG(struct View *, struct ViewPort *)
ML: d0 = CalcIVG(a0, a1)
Arguments: view = View structure
viewPort = ViewPort in question
Result: count = number of scan lines needed to execute all the ViewPort's copper list instructions; zero if unsuccessful

CallHookPkt (Revision 2.0)

Description: invokes a Hook callback function
Library: utility.library
Offset: -$66
Syntax: return = CallHookPkt(hook, object, message)
C: ULONG CallHookPkt(struct Hook *, APTR, APTR)
ML: d0 = CallHookPkt(a0, a2, a1)
Arguments: hook = an initialized Hook structure
object = user-defined data structure meaningful to the function being called
message = the message to pass to the callback function
Result: return = the value returned by the callback function

Cause

Description: causes a software interrupt
Library: exec.library
Offset: -$B4
Syntax: Cause(interrupt)
C: VOID Cause(struct Interrupt *)
ML: Cause(a1)
Arguments: interrupt = pointer to a properly initialized interrupt node
Result: none

CBump

Description: increments user copper list pointer
Library: graphics.library
Offset: -$16E
Syntax: CBump(c)

C: VOID CBump(struct UCopList *)
ML: CBump(a1)
Arguments: c = UCopList structure
Result: none

ChangeExtSpriteA (Revision 3.0)

Description: changes the pointer sprite's image
Library: graphics.library
Offset: -$402
Syntax: success = ChangeExtSpriteA(viewPort, oldSprite, newSprite, tags)
C: BOOL = ChangeExtSpriteA(struct ViewPort *, struct ExtSprite *, struct ExtSprite *, struct TagList *)
ML: d0 = ChangeExtSpriteA(a0, a1, a2, a3)
Arguments: viewPort = ViewPort structure to which this sprite is relative
oldsprite = old ExtSprite structure
newsprite = new ExtSprite structure
tags = tag list of attributes:
SPRITEA_Width ($81000000), SPRITEA_XReplication ($81000002), SPRITEA_YReplication ($81000004), SPRITEA_OutputHeight ($81000006), SPRITEA_Attached ($81000008), SPRITEA_OldDataFormat ($8100000A)
Result: success = zero if unsuccessful

ChangeMode (Revision 2.0)

Description: changes the current mode of a lock or filehandle
Library: dos.library
Offset: -$1C2
Syntax: success = ChangeMode(type, object, newmode)
C: BOOL ChangeMode(ULONG, BPTR, ULONG)
ML: d0 = ChangeMode(d1, d2, d3)
Arguments: type = what to change—CHANGE_LOCK ($00000000), CHANGE_FH ($00000001)
object = a lock or filehandle
newmode = the new mode
Result: success = zero if unsuccessful

ChangeScreenBuffer (Revision 3.0)

Description: swaps the bitmaps in a double-buffered screen
Library: intuition.library
Offset: -$30C
Syntax: success = ChangeScreenBuffer(screen, screenBuffer)
C: ULONG ChangeScreenBuffer(struct Screen *, struct ScreenBuffer *)
ML: d0 = ChangeScreenBuffer(a0, a1)
Arguments: screen = double buffered screen
ScreenBuffer = pointer returned by AllocScreenBuffer( ) function
Result: success = zero if unsuccessful

ChangeSprite

Description: changes the shape of a sprite
Library: graphics.library
Offset: -$1A4
Syntax: ChangeSprite(viewPort, simpleSprite, newData)
C: VOID = ChangeExtSprite(struct ViewPort *, struct SimpleSprite *, short *)
ML: ChangeSprite(a0, a1, a2)
Arguments: viewPort = ViewPort structure
simpleSprite = initialized SimpleSprite structure
newData = sprite definition
Result: none

ChangeVPBitMap (Revision 3.0)

Description: swaps bitmaps in a double-buffered display
Library: graphics.library
Offset: -$3AE
Syntax: ChangeVPBitMap(viewPort, bitMap, db)
C: VOID ChangeVPBitMap(struct ViewPort *, struct BitMap *, struct DBufInfo *)
ML: ChangeVPBitMap(a0, a1, a2)
Arguments: viewPort = ViewPort structure for double-buffered display
bitMap = BitMap structure to make visible
db = DBufInfo structure
Result: none

ChangeWindowBox (Revision 2.0)

Description: modifies a window's size and position
Library: intuition.library
Offset: -$1E6
Syntax: ChangeWindowBox(window, left, top, width, height)
C: VOID ChangeWindowBox(struct Window *, WORD, WORD, WORD, WORD)
ML: ChangeWindowBox(a0, d0, d1, d2, d3)
Arguments: window = window to change
left, top = new window position
width, height = new window size
Result: none

CheckDate (Revision 2.0)

Description: tests the validity of a ClockData structure
Library: utility.library
Offset: -$84
Syntax: seconds = CheckDate(date)
C: ULONG CheckDate(struct ClockData *)
ML: d0 = CheckDate(a0)
Arguments: date = an initialized ClockData structure
Result: seconds = zero if the date is invalid;
number of seconds from January 1, 1978 to date if valid

CheckIO

Description: returns the status of an IORequest
Library: exec.library
Offset: -$1D4
Syntax: result = CheckIO(iORequest)
C: struct IORequest *CheckIO(struct IORequest *)
ML: d0 = CheckIO(a1)
Arguments: iORequest = the IORequest block to check
Result: result = the IORequest block; zero if the I/O request is still in progress

CheckSignal (Revision 2.0)

Description: checks for break signals
Library: dos.library
Offset: -$318
Syntax: signals = CheckSignal(mask)
C: ULONG CheckSignals(ULONG)
ML: d0 = CheckSignal(d1)
Arguments: mask = signals to check for
Result: signals = signals that will be checked

ClearCxObjError (Revision 2.0)

Description: clears the accumulated error value of a commodity
Library: commodities.library
Offset: -$48
Syntax: ClearCxObjError(cxobj)
C: VOID ClearCxObjError(CxObj *)
ML: ClearCxObjError(a0)
Arguments: cxobj = commodity object to modify
Result: none

ClearDMRequest

Description: removes the DMRequest of a window
Library: intuition.library
Offset: -$30
Syntax: response = ClearDMRequest(window)
C: BOOL ClearDMRequest(struct Window *)
ML: d0 = ClearDMRequest(a0)
Arguments: window = window with unwanted DMRequest
Result: response = zero if unsuccessful

ClearEOL

Description: clears from the current position to end of the line
Library: graphics.library
Offset: -$2A

Syntax: ClearEOL(rastPort)
C: VOID ClearEOL(struct RastPort *)
ML: ClearEOL(a1)
Arguments: rastPort = RastPort structure
Result: none

ClearMenuStrip
Description: removes a window's menu strip
Library: intuition.library
Offset: -$36
Syntax: ClearMenuStrip(window)
C: VOID ClearMenuStrip(struct Window *)
ML: ClearMenuStrip(a0)
Arguments: window = window with unwanted menu
Result: none

ClearPointer
Description: returns the mouse pointer to its default shape
Library: intuition.library
Offset: -$3C
Syntax: ClearPointer(window)
C: VOID ClearPointer(struct Window *)
ML: ClearPointer(a0)
Arguments: window = window with unwanted custom mouse-pointer definition
Result: none

ClearRectRegion
Description: clears a rectangular area of a region
Library: graphics.library
Offset: -$20A
Syntax: success = ClearRectRegion(region, rectangle)
C: BOOL ClearRectRegion(struct Region *, struct Rectangle *)
ML: d0 = ClearRectRegion(a0, a1)
Arguments: region = Region structure
rectangle = Rectangle structure defining region to clear
Result: success = zero if unsuccessful

ClearRegion
Description: removes all rectangles from region
Library: graphics.library
Offset: -$210
Syntax: ClearRegion(region)
C: viod ClearRegion(struct Region *)
ML: ClearRegion(a0)

Arguments: region = Region structure
Result: none

ClearRexxMsg (Revision 2.0)
Description: releases and clears the arguments in a RexxMsg
Library: rexxsyslib.library
Offset: -$9C
Syntax: ClearRexxMsg(msgptr, count)
C: VOID ClearRexxMsg(struct RexxMsg *, ULONG)
ML: ClearRexxMsg(a0, d0)
Arguments: msgptr = a RexxMsg
count = number of slots to be cleared
Result: none

ClearScreen
Description: clears from current position to end of RastPort
Library: graphics.library
Offset: -$30
Syntax: ClearScreen(rastPort)
C: VOID ClearScreen(struct RastPort *)
ML: ClearScreen(a1)
Arguments: rastPort = RastPort structure
Result: none

Cli (Revision 2.0)
Description: gets a pointer to the CLI structure of the current process
Library: dos.library
Offset: -$1EC
Syntax: cli_ptr = Cli( )
C: struct CommandLineInterface *Cli(VOID)
ML: d0 = Cli( )
Arguments: none
Result: cli_ptr = the CLI structure; zero if unsuccessful

CliInitNewcli (Revision 2.0)
Description: sets up a shell process
Library: dos.library
Offset: -$3A2
Syntax: flags = CliInitNewcli(packet)
C: LONG CliInitNewcli(struct DosPacket *)
ML: d0 = CliInitNewcli(a0)
Arguments: packet = initial packet that was sent to your process's MsgPort

Result: flags = bit flags; $00000001 = RUN provided output stream, $00000002 = user provided input stream, $00000004 = System( ) call, $00000008 = an asynch call, $80000000 = previous flags are valid, otherwise an error occured and this long value is a pointer to your process

CliInitRun (Revision 2.0)

Description: sets up a detachable shell process
Library: dos.library
Offset: -$3A8
Syntax: flags = CliInitRun(packet)
C: LONG CliInitRun(struct DosPacket *)
ML: d0 = CliInitRun(a0)
Arguments: packet = initial packet that was sent to your process's MsgPort
Result: flags = bit flags; $00000001 = RUN provided output stream,
$00000002 = user provided input stream,
$00000004 = System( ) call,
$00000008 = an asynch call,
$80000000 = previous flags are valid, otherwise an error occured and this long value is a pointer to your process

ClipBlit

Description: copies (blits) data from one RastPort to another using BltBitMap( ), but avoids trashing overlapping layers
Library: graphics.library
Offset: -$228
Syntax: ClipBlit(srcRastPort, srcX, srcY, destRastPort, destX, destY, sizeZ, sizeY, minterm)
C: VOID ClipBlit(struct RastPort *, WORD, WORD, struct RastPort *, WORD, WORD, WORD, WORD, UBYTE)
ML: ClipBlit(a0, d0, d1, a1, d2, d3, d4, d5, d6)
Arguments: srcRastPort = source RastPort for blit
srcX, srcY = upperleft corner of source rectangle
destRastPort = destination RastPort for blit
destX, destY = upper left corner of destination rectangle
sizeX, sizeY = width and height of rectangle; must be at least 1 by 1
minterm = blitter logic function to apply to rectangle
Result: none

CloneTagItems (Revision 2.0)

Description: copies a tag list
Library: utility.library
Offset: -$48
Syntax: clone = CloneTagItems(original)
C: struct TagItem *CloneTagItems(struct TagItem *)
ML: d0 = CloneTagItems(a0)

Arguments: original = tag list to copy
Result: clone = copy of the original tag list; zero if unsuccessful

Close
Description: closes an open file
Library: dos.library
Offset: -$24
Syntax: success = Close(fh)
C: BOOL Close(BPTR)
ML: d0 = Close(d1)
Arguments: fh = filehandle of file to close
Result: success = zero if unsuccessful (return value valid only in Revision 2.0)

CloseAmigaGuide (Revision 3.0)
Description: closes a client
Library: amigaguide.library
Offset: -$42
Syntax: CloseAmigaGuide(handle)
C: VOID CloseAmigaGuide(AMIGAGUIDECONTEXT)
ML: CloseAmigaGuide(a0)
Arguments: handle = handle to an AmigaGuide system
Result: none

CloseCatalog (Revision 3.0)
Description: closes a message catalog
Library: locale.library
Offset: -$24
Syntax: CloseCatalog(catalog)
C: VOID CloseCatalog(struct Catalog *)
ML: CloseCatalog(a0)
Arguments: catalog = Catalog structure returned by OpenCatalog( )
Result: none

CloseClipboard (Revision 2.0)
Description: closes and frees an open ClipboardHandle
Library: iffparse.library
Offset: -$FC
Syntax: CloseClipboard(clipHandle)
C: VOID CloseClipboard(struct ClipboardHandle *)
ML: CloseClipboard(a0)
Arguments: clipHandle = a ClipboardHandle structure returned by OpenClipboard( )
Result: none

CloseDevice
Description: concludes access to a device
Library: exec.library
Offset: -$1C2
Syntax: CloseDevice(iORequest)
C: VOID CloseDevice(struct IORequest *)
ML: CloseDevice(a1)
Arguments: iORequest = an IORequest structure
Result: none

CloseEngine (Revision 3.0)
Description: closes an engine
Library: bullet.library
Offset: -$24
Syntax: CloseEngine(handle)
C: VOID CloseEngine(struct GlyphEngine *)
ML: CloseEngine(a0)
Arguments: handle = handle returned by OpenEngine( )
Result: none

CloseFont
Description: releases a system font
Library: graphics.library
Offset: -$4E
Syntax: CloseFont(font)
C: VOID CloseFont(struct TextFont *)
ML: CloseFont(a1)
Arguments: font = a Font as returned by the OpenFont( ) or OpenDiskFont( ) functions
Result: none

CloseIFF (Revision 2.0)
Description: closes an IFF context
Library: iffparse.library
Offset: -$30
Syntax: CloseIFF(iff)
C: VOID CloseIFF(struct IFFHandle *)
ML: CloseIFF(a0)
Arguments: iff = an IFFHandle structure returned by OpenIFF( )
Result: none

CloseLibrary
Description: concludes access to a library
Library: exec.library
Offset: -$19E

Syntax: CloseLibrary(library)
C: VOID CloseLibrary(struct Library *)
ML: CloseLibrary(a1)
Arguments: library = the library node to close
Result: none

CloseLocale (Revision 3.0)

Description: closes a locale
Library: locale.library
Offset: -$2A
Syntax: CloseLocale(locale)
C: VOID CloseLocale(struct Locale *)
ML: CloseLocale(a0)
Arguments: locale = Locale structure returned by OpenLocale( )
Result: none

CloseMonitor (Revision 2.0)

Description: closes a MonitorSpec
Library: graphics.library
Offset: -$2D0
Syntax: error = CloseMonitor(monitorSpec)
C: LONG CloseMonitor(struct MonitorSpec *)
ML: d0 = CloseMonitor(a0)
Arguments: monitorSpec = a MonitorSpec as returned by the OpenMonitor( ) function
Result: error = zero if successful

CloseScreen

Description: closes an Intuition screen
Library: intuition.library
Offset: -$42
Syntax: success = CloseScreen(screen)
C: BOOL CloseScreen(struct Screen *)
ML: d0 = CloseScreen(a0)
Arguments: screen = unwanted screen
Result: success = zero if unsuccessful (valid only under Revision 2.0)

CloseWindow

Description: closes an Intuition window
Library: intuition.library
Offset: -$48
Syntax: CloseWindow(window)
C: VOID CloseWindow(struct Window *)
ML: CloseWindow(a0)
Arguments: window = unwanted window
Result: none

CloseWorkBench

Description: closes the Workbench screen
Library: intuition.library
Offset: -$4E
Syntax: success = CloseWorkBench( )
C: LONG CloseWorkBench(VOID)
ML: d0 = CloseWorkBench( )
Arguments: none
Result: success = zero if unsuccessful

CoerceMode (Revision 3.0)

Description: calculates ViewPort mode coercion
Library: graphics.library
Offset: -$3A8
Syntax: id = CoerceMode(realViewPort, monitorID, flags);
C: ULONG CoerceMode(struct ViewPort *, ULONG, ULONG)
ML: d0 = CoerceMode(a0, d0, d1)
Arguments: realViewPort = ViewPort to coerce
monitorID = Monitor number to coerce—a mode masked with MONITOR_ID_MASK ($FFFF1000)
flags = PRESERVE_COLORS ($00000001), AVOID_FLICKER ($00000002)
Result: id = ModeID of the best mode to coerce to; -1 if unsuccessful

ColdReboot (Revision 2.0)

Description: reboots the Amiga
Library: exec.library
Offset: -$2D6
Syntax: ColdReboot( )
C: VOID ColdReboot(VOID)
ML: ColdReboot( )
Arguments: a chaotic pile of disoriented bits
Result: an altogether totally-integrated living system

CollectionChunk (Revision 2.0)

Description: declares a chunk type for collection
Library: iffparse.library
Offset: -$8A
Syntax: error = CollectionChunk(iff, type, id)
C: LONG CollectionChunk(struct IFFHandle *, LONG, LONG)
ML: d0 = CollectionChunk(a0, d0, d1)
Arguments: iff = an IFFHandle structure; not necessarily open
type = type code for the chunk to declare
id = identifier for the chunk to declare
Result: error = zero if successful; nonzero error-code if unsuccessful

CollectionChunks (Revision 2.0)

Description: declares multiple collection chunks
Library: iffparse.library
Offset: -$90
Syntax: error = CollectionChunks(iff, propArray, numPairs)
C: LONG CollectionChunks(struct IFFHandle *, LONG *, LONG)
ML: d0 = CollectionChunks(a0, a1, d0)
Arguments: iff = an IFFHandle structure; not necessarily open
propArray = array of chunk types and identifiers
numPairs = number of pairs in array
Result: error = zero if successful; nonzero error-code if unsuccessful

CompareDates (Revision 2.0)

Description: compares two datestamps
Library: dos.library
Offset: -$2E2
Syntax: result = CompareDates(date1, date2)
C: LONG CompareDates(struct DateStamp *,struct DateStamp *)
ML: d0 = CompareDates(d1, d2)
Arguments: date1, date2 = DateStamps to compare
Result: result = negative if date1 is later than date2; 0 if same date;
positive if date2 is later than date1

ConfigBoard

Description: configures an expansion board
Library: expansion.library
Offset: -$3C
Syntax: ConfigBoard(board, configDev)
C: VOID ConfigBoard(APTR, struct ConfigDev *)
ML: ConfigBoard(a0, a1)
Arguments: board = current address for the expansion board
configDev = initialized ConfigDev structure returned by AllocConfigDev( )
Result: none

ConvToLower (Revision 3.0)

Description: converts a character to lower case based on a locale
Library: locale.library
Offset: -$30
Syntax: char = ConvToLower(locale, character)
C: ULONG ConvToLower(struct Locale *, ULONG)
ML: d0 = ConvToLower(a0, d0)
Arguments: locale = the locale to use for this conversion
character = the character to convert
Result: char = the converted character

ConvToUpper (Revision 3.0)
Description: converts a character to upper case based on a locale
Library: locale.library
Offset: -$36
Syntax: char = ConvToUpper(locale, character)
C: ULONG ConvToUpper(struct Locale *, ULONG)
ML: d0 = ConvToUpper(a0, d0)
Arguments: locale = the locale to use for this conversion
character = the character to convert
Result: char = the converted character

CopyMem
Description: performs general purpose memory copies
Library: exec.library
Offset: -$270
Syntax: CopyMem(source, dest, size)
C: VOID CopyMem(APTR, APTR, ULONG)
ML: CopyMem(a0, a1, d0)
Arguments: source = the source data region
dest = the destination data region
size = the size of the memory copy in bytes
Result: none

CopyMemQuick
Description: performs an optimized memory copy
Library: exec.library
Offset: -$276
Syntax: CopyMemQuick(source, dest, size)
C: VOID CopyMemQuick(ULONG *, ULONG *, ULONG)
ML: CopyMemQuick(a0, a1, d0)
Arguments: source = the source data region, longword-aligned
dest = the destination data region, longword-aligned
size = the size of the memory area in bytes
Result: none

CopySBitMap
Description: copies all bits from a SuperBitMap to Layer bounds
Library: graphics.library
Offset: -$1C2
Syntax: CopySBitMap(layer)
C: VOID CopySBitMap(struct Layer *)
ML: CopySBitMap(a0)
Arguments: layer = a locked SuperBitMap Layer
Result: none

CreateArgstring (Revision 2.0)

Description: allocates and fills an argument string for an ARexx message
Library: rexxsyslib.library
Offset: -$7E
Syntax: argstr = CreateArgstring(string, length)
C: UBYTE *CreateArgstring(UBYTE *, ULONG)
ML: d0,a0 = CreateArgstring(a0, d0)
Arguments: string = the input string
length = number of bytes of the input string to copy
Result: argstr = the newly-created argument string

CreateBehindHookLayer (Revision 2.0)

Description: creates a new layer behind all existing layers with callback
Library: layers.library
Offset: -$C0
Syntax: result = CreateBehindHookLayer(li, bm, x0, y0, x1, y1, flags, hook [,bm2])
C: struct Layer *CreateBehindHookLayer(struct Layer_Info *, struct BitMap *, LONG, LONG, LONG, LONG, LONG, struct Hook * [, struct Bitmap *])
ML: d0 = CreateBehindHookLayer(a0, a1, d0, d1, d2, d3, d4, a3 [, a2])
Arguments: li = a LayerInfo structure
bm = a common BitMap used by all Layers
x0 = left coordinate of layer
y0 = top coordinate of layer
x1 = right coordinate of layer
y1 = bottom coordinate of layer
flags = specifies type of layer:
LAYERSIMPLE ($00000001), LAYERSMART ($00000002), LAYERSUPER ($00000004), LAYERUPDATING ($00000010), LAYERBACKDRO ($00000040), LAYERREFRESH ($00000080), LAYERIREFRESH ($00000200), LAYERIREFRESH2 ($00000400), LAYER_CLIPRECTS_LOST ($00000100)
hook = callback hook function for layer
bm2 = an optional Super BitMap
Result: result = a new Layer structure; zero if unsuccessful

CreateBehindLayer

Description: creates a new layer behind all existing layers
Library: layers.library
Offset: -$2A
Syntax: result = CreateBehindLayer(li, bm, x0, y0, x1, y1, flags [,bm2])
C: struct Layer *CreateBehindLayer(struct Layer_Info *, struct BitMap *, LONG, LONG, LONG, LONG, LONG [, struct Bitmap *])
ML: d0 = CreateBehindLayer(a0, a1, d0, d1, d2, d3, d4 [, a2])

Arguments: li = a LayerInfo structure
bm = a common BitMap used by all Layers
x0 = left coordinate of layer
y0 = top coordinate of layer
x1 = right coordinate of layer
y1 = bottom coordinate of layer
flags = specifies type of layer—LAYERSIMPLE ($00000001), LAYERSMART ($00000002), LAYERSUPER ($00000004), LAYERUPDATING ($00000010), LAYERBACKDROP ($00000040), LAYERREFRESH ($00000080), LAYERIREFRESH ($00000200), LAYERIREFRESH2 ($00000400), LAYER_CLIPRECTS_LOST ($00000100)
Result: result = a new Layer structure; zero if unsuccessful

CreateContext (Revision 2.0)

Description: creates memory for GadTools context data
Library: gadtools.library
Offset: -$72
Syntax: gad = CreateContext(glistpointer);
C: struct Gadget *CreateContext(struct Gadget **)
ML: d0 = CreateContext(a0)
Arguments: glistpointer = address of a Gadget structure pointer which has been set to zero
Result: gad = pointer to context gadget; zero if unsuccessful

CreateCxObj (Revision 2.0)

Description: allocates a commodity object
Library: commodities.library
Offset: -$1E
Syntax: cxobj = CreateCxObj(type, arg1, arg2)
C: CxObj *CreateCxObj(ULONG, LONG, LONG)
ML: d0 = CreateCxObj(d0, a0, a1)
Arguments: type = type of object to allocate:
CX_FILTER ($00000001), CX_TYPEFILTER ($00000002), CX_SEND ($00000003), CX_SIGNAL ($00000004), CX_TRANSLATE ($00000005), CX_BROKER ($00000006), CX_DEBUG ($00000007), CX_CUSTOM ($00000008), CX_ZERO ($00000009)
arg1 = type-dependent data
arg2 = type-dependant data
Result: cxobj = new commodities object; zero if unsuccessful

CreateDir

Description: creates a new directory
Library: dos.library
Offset: -$78
Syntax: lock = CreateDir(name)
C: BPTR CreateDir(STRPTR)

ML: d0 = CreateDir(d1)
Arguments: name = directory name
Result: lock = BCPL pointer to a lock; zero if unsuccessful

CreateGadgetA (Revision 2.0)

Description: allocates and initializes a GadTools gadget
Library: gadtools.library
Offset: -$1E
Syntax: gad = CreateGadgetA(kind, previous, newgad, tags)
C: struct Gadget *CreateGadgetA(ULONG, struct Gadget *, struct NewGadget *, struct TagItem *)
ML: d0 = CreateGadgetA(d0, a0, a1, a2)
Arguments: kind = type of gadget to create:
GENERIC_KIND ($00000000), BUTTON_KIND ($00000001), CHECKBOX_KIND ($00000002), INTEGER_KIND ($00000003), LISTVIEW_KIND ($00000004), MX_KIND ($00000005), NUMBER_KIND ($00000006), CYCLE_KIND ($00000007), PALETTE_KIND ($00000008), SCROLLER_KIND ($00000009), SLIDER_KIND ($0000000B), STRING_KIND ($0000000C), TEXT_KIND ($0000000D)
previous = gadget to which the new gadget should be attached
newgad = initialized NewGadget structure specifying attributes of the gadget
tags = tag list of attributes—see <libraries/gadtools.h> for description
Result: gad = pointer to the new gadget; zero if unsuccessful

CreateIORequest (Revision 2.0)

Description: creates an IORequest structure
Library: exec.library
Offset: -$28E
Syntax: ioReq = CreateIORequest(ioReplyPort, size)
C: struct IORequest *CreateIORequest(struct MsgPort *, ULONG)
ML: d0 = CreateIORequest(a0, d0)
Arguments: ioReplyPort = a port for replies (an initialized message port, as created by CreateMsgPort( )); if 0, this function fails
size = the size of the I/O request to create
Result: ioReq = the new IORequest block; zero if unsuccessful

CreateMenusA (Revision 2.0)

Description: allocates and initializes a menu structure
Library: gadtools.library
Offset: -$30
Syntax: menu = CreateMenusA(newmenu, tags)
C: struct Menu *CreateMenusA(struct NewMenu *, struct TagItem *)
ML: d0 = CreateMenusA(a0, a1)

Arguments: newmenu = array of initialized NewMenu structures
tags = tag list of attributes—GTMN_TextAttr ($80080031), GTMN_FrontPen ($80080032), GTMN_FullMenu ($8008003E), GTMN_SecondaryError ($8008003F), Revision 3.0: GTMN_Checkmark ($80080041), GTMN_AmigaKey ($80080042), GTMN_NewLookMenus ($80080043)
Result: menu = initialized menu structure; zero if unsuccessful

CreateMsgPort (Revision 2.0)

Description: allocates and initializes a new message port
Library: exec.library
Offset: -$29A
Syntax: msgPort = CreateMsgPort( )
C: struct MsgPort *CreateMsgPort(VOID)
ML: CreateMsgPort( )
Arguments: none
Result: msgPort = a new MsgPort structure ready for use; zero if out of memory or signals
If you wish to add this port to the public port list, fill in the ln_Name and ln_Pri fields, then call AddPort( ). Don't forget RemPort( )!

CreateNewProc (Revision 2.0)

Description: creates a new process
Library: dos.library
Offset: -$1F2
Syntax: process = CreateNewProc(tags)
C: struct Process *CreateNewProcTags(Strict TagItem*)
ML: d0 = CreateNewProc(d1)
Arguments: tags = tag list specifying additional attributes—see <dos/dostags.h> for description
Result: process = the new process; zero if unsuccessful

CreatePool (Revision 3.0)

Description: generates a private memory pool header
Library: exec.library
Offset: -$2B8
Syntax: newPool = CreatePool(memFlags, puddleSize, threshSize)
C: VOID *CreatePool(ULONG, ULONG, ULONG)
ML: d0 = CreatePool(d0, d1, d2)
Arguments: memFlags = a memory flags specifier, as taken by AllocMem( )
puddleSize = the size of puddles
threshSize = the largest allocation that goes into normal puddles; this *must* be less than or equal to puddleSize (CreatePool( ) will fail if it is not)
Result: newPool = a new pool header; zero if unsuccessfull

CreateProc

Description: creates a new process
Library: dos.library
Offset: -$8A
Syntax: process = CreateProc(name, pri, seglist, stackSize)
C: struct MsgPort *CreateProc(STRPTR, LONG, BPTR, LONG)
ML: d0 = CreateProc(d1, d2, d3, d4)
Arguments: name = process name
pri = process priority; -128 to +127
seglist = seglist of new process
stackSize = stack size for process; must be an even multiple of 4
Result: process = the new process' msgport; zero if unsuccessful

CreateRexxMsg (Revision 2.0)

Description: allocates an ARexx message structure
Library: rexxsyslib.library
Offset: -$90
Syntax: rexxmsg = CreateRexxMsg(port, extension, host)
C: struct RexxMsg *CreateRexxMsg(struct MsgPort *, UBYTE *, UBYTE *)
ML: d0 = CreateRexxMsg(a0, a1, d0)
Arguments: port = a valid message port to receive the reply to this message
extension = text string used as filename extension for REXX scripts;
0 indicates default of "REXX"
host = name of the default host port; 0 indicates default of "REXX"
Result: rexxmsg = RexxMsg structure; zero if unsuccessful

CreateUpfrontHookLayer (Revision 2.0)

Description: creates a new layer in front of all existing layers with callback
Library: layers.library
Offset: -$BA
Syntax: result = CreateUpfrontHookLayer(li, bm, x0, y0, x1, y1, flags, hook [,bm2])
C: struct Layer *CreateUpfrontHookLayer(struct Layer_Info *, struct BitMap *, LONG, LONG, LONG, LONG, LONG, struct Hook * [, struct Bitmap *])
ML: d0 = CreateUpfrontHookLayer(a0, a1, d0, d1, d2, d3, d4, a3 [, a2])
Arguments: li = a LayerInfo structure
bm = a common BitMap used by all Layers
x0 = left coordinate of layer
y0 = top coordinate of layer
x1 = right coordinate of layer
y1 = bottom cooordinate of layer:
flags = specifies type of layer
LAYERSIMPLE ($00000001), LAYERSMART ($00000002),
LAYERSUPER ($00000004), LAYERUPDATING ($00000010),
LAYERBACKDROP ($00000040), LAYERREFRESH
($00000080), LAYERIREFRESH ($00000200),

LAYERIREFRESH2 ($00000400), LAYER_CLIPRECTS_LOST ($00000100)
hook = callback hook function for layer
Result: result = a new Layer structure; zero if unsuccessful

CreateUpfrontLayer
Description: creates a new layer behind all existing layers
Library: layers.library
Offset: -$24
Syntax: result = CreateUpfrontLayer(li, bm, x0, y0, x1, y1, flags [,bm2])
C: struct Layer *CreateUpfrontLayer(struct Layer_Info *, struct BitMap *, LONG, LONG, LONG, LONG, LONG [, struct Bitmap *])
ML: d0 = CreateUpfrontLayer(a0, a1, d0, d1, d2, d3, d4 [, a2])
Arguments: li = a LayerInfo structure
bm = a common BitMap used by all Layers
x0 = left coordinate of layer
y0 = top coordinate of layer
x1 = right coordinate of layer
y1 = bottom coordinate of layer
flags = specifies type of layer:
LAYERSIMPLE ($00000001), LAYERSMART ($00000002), LAYERSUPER ($00000004), LAYERUPDATING ($00000010), LAYERBACKDROP ($00000040), LAYERREFRESH ($00000080), LAYERIREFRESH ($00000200), LAYERIREFRESH2 ($00000400), LAYER_CLIPRECTS_LOST ($00000100)
Result: result = a new Layer structure; zero if unsuccessful

CurrentChunk (Revision 2.0)
Description: retrieves context node for current chunk
Library: iffparse.library
Offset: -$AE
Syntax: top = CurrentChunk(iff)
C: struct ContextNode *CurrentChunk(struct IFFHandle *)
ML: d0 = CurrentChunk(a0)
Arguments: iff = an IFFHandle structure
Result: top = the top context node; zero if unsuccessful

CurrentDir
Description: changes the current directory
Library: dos.library
Offset: -$7E
Syntax: oldLock = CurrentDir(lock)
C: BPTR CurrentDir(BPTR)
ML: d0 = CurrentDir(d1)
Arguments: lock = a directory lock
Result: oldLock = previous current-directory lock

CurrentTime

Description: returns current system time
Library: intuition.library
Offset: -$54
Syntax: CurrentTime(seconds, micros)
C: VOID CurrentTime(ULONG *, ULONG *)
ML: CurrentTime(a0, a1)
Arguments: seconds = a ULONG variable to receive the current seconds value
micros = a ULONG variable for the current microseconds value
Result: none

CxBroker (Revision 2.0)

Description: creates a commodity broker
Library: commodities.library
Offset: -$24
Syntax: broker = CxBroker(nb, error)
C: CxObj *CxBroker(struct NewBroker *, LONG *)
ML: d0 = CxBroker(a0, d0)
Arguments: nb = initialized NewBroker structure
error = buffer to store error code—CBERR_OK ($00000000), CBERR_SYSERR ($00000001), CBERR_DUP ($00000002), CBERR_VERSION ($00000003)
Result: broker = new broker object; zero if unsuccessful

CxMsgData (Revision 2.0)

Description: accesses a commodity message's data area
Library: commodities.library
Offset: -$90
Syntax: data = CxMsgData(cxm)
C: APTR CxMsgData(struct CxMsg *)
ML: d0 = CxMsgData(a0)
Arguments: cxm = commodity message to access
Result: data = pointer to the message's data; zero if unsuccessful

CxMsgID (Revision 2.0)

Description: obtains a commodity message's ID
Library: commodities.library
Offset: -$96
Syntax: id = CxMsgID(cxm)
C: LONG CxMsgID(struct CxMsg *)
ML: d0 = CxMsgID(a0)
Arguments: cxm = commodity message to access
Result: id = ID value of the commodity message

CxMsgType (Revision 2.0)

Description: obtains a commodity message's type
Library: commodities.library
Offset: -$8A
Syntax: type = CxMsgType(cxm)
C: ULONG CxMsgType(struct CxMsg *)
ML: d0 = CxMsgType(a0)
Arguments: cxm = commodity message to access
Result: type = type of the commodity message—CXM_IEVENT ($00000010), CXM_COMMAND ($00000020)

CxObjError (Revision 2.0)

Description: obtains a commodity object's accumulated error
Library: commodities.library
Offset: -$42
Syntax: error = CxObjError(cxobj)
C: LONG CxObjError(CxObj *)
ML: d0 = CxObjError(a0)
Arguments: cxobj = commodity object to access
Result: error = accumulated error of the commodity object: COERR_ISNULL ($00000001), COERR_NULLATTACH ($00000002), COERR_BADFILTER ($00000004), COERR_BADTYPE ($00000008)

CxObjType (Revision 2.0)

Description: obtains a commodity object's type
Library: commodities.library
Offset: -$3C
Syntax: type = CxObjType(cxobj)
C: ULONG CxObjType(CxObj *)
ML: d0 = CxObjType(a0)
Arguments: cxobj = commodity object to access
Result: type = type of the commodity object—CX_FILTER ($00000001), CX_TYPEFILTER ($00000002), CX_SEND ($00000003), CX_SIGNAL ($00000004), CX_TRANSLATE ($00000005), CX_BROKER ($00000006), CX_DEBUG ($00000007), CX_CUSTOM ($00000008), CX_ZERO ($00000009)

Date2Amiga (Revision 2.0)

Description: calculates seconds from January 1, 1978
Library: utility.library
Offset: -$7E
Syntax: seconds = Date2Amiga(date)
C: ULONG Date2Amiga(struct ClockData *)
ML: d0 = Date2Amiga(a0)
Arguments: date = an initialized ClockData structure
Result: seconds = the number of seconds from January 1, 1978 to date

DateStamp

Description: obtains the date and time in internal format
Library: dos.library
Offset: -$C0
Syntax: ds = DateStamp(ds);
C: struct DateStamp *DateStamp(struct DateStamp *)
ML: d0 = DateStamp(d1)
Arguments: ds = DateStamp structure to hold current date
Result: ds = current date

DateToStr (Revision 2.0)

Description: converts a DateStamp to a string
Library: dos.library
Offset: -$2E8
Syntax: success = DateToStr(datetime)
C: BOOL DateToStr(struct DateTime *)
ML: d0 = DateToStr(d1)
Arguments: DateTime = an initialized DateTime structure
Result: success = zero if unsuccessful

Deallocate

Description: deallocates a block of memory
Library: exec.library
Offset: -$C0
Syntax: Deallocate(memHeader, memoryBlock, byteSize)
C: VOID Deallocate(struct MemHeader *, APTR, ULONG)
ML: Deallocate(a0, a1, d0)
Arguments: memHeader = the memory header which oversees this block
memoryBlock = the memory block to free
byteSize = the size of the block in bytes; may be 0
Result: none

Debug

Description: runs the system debugger
Library: exec.library
Offset: -$72
Syntax: Debug(flags)
C: VOID Debug(ULONG)
ML: Debug(d0)
Arguments: none
Result: none

Delay

Description: pauses a process for a specified time
Library: dos.library

Offset: -$C6
Syntax: Delay(ticks)
C: VOID Delay(ULONG)
ML: Delay(d1)
Arguments: ticks = number of ticks (50ths of a second) to pause
Result: none

DeleteArgstring (Revision 2.0)
Description: frees argument string allocated by CreateArgstring( )
Library: rexxsyslib.library
Offset: -$84
Syntax: DeleteArgstring(argstring)
C: VOID DeleteArgstring(UBYTE *)
ML: DeleteArgstring(a0)
Arguments: argstring = argument string returned by CreatArgstring( )
Result: none

DeleteCxObj (Revision 2.0)
Description: frees a commmodity object
Library: commodities.library
Offset: -$30
Syntax: DeleteCxObj(cxobj)
C: VOID DeleteCxObj(CxObj *)
ML: DeleteCxObj(a0)
Arguments: cxobj = commodity object to free
Result: none

DeleteCxObjAll (Revision 2.0)
Description: frees a tree of commodity objects
Library: commodities.library
Offset: -$36
Syntax: DeleteCxObjAll(cxobj)
C: VOID DeleteCxObjAll(CxObj *)
ML: DeleteCxObjAll(a0)
Arguments: cxobj = first commodity object in tree
Result: none

DeleteDiskObject (Revision 2.0)
Description: deletes a Workbench disk object from disk
Library: icon.library
Offset: -$8A
Syntax: success = DeleteDiskObject(name)
C: BOOL DeleteDiskObject(char *)
ML: d0 = DeleteDiskObject(a0)
Arguments: name = name of the object to delete
Result: success = zero if unsuccessful

DeleteFile

Description: deletes a file or directory
Library: dos.library
Offset: -$48
Syntax: success = DeleteFile(name)
C: BOOL DeleteFile(STRPTR)
ML: d0 = DeleteFile(d1)
Arguments: name = name of file or directory to delete
Result: success = zero if unsuccessful

DeleteIORequest (Revision 2.0)

Description: frees a request made by CreateIORequest( )
Library: exec.library
Offset: -$294
Syntax: DeleteIORequest(ioReq);
C: VOID DeleteIORequest(struct IORequest *)
ML: DeleteIORequest(a0)
Arguments: ioReq = the IORequest block to be freed; may be 0
Result: none
This function uses the mn_Length field to determine how much memory to free.

DeleteLayer

Description: deletes a layer from the layer list
Library: layers.library
Offset: -$5A
Syntax: success = DeleteLayer(dummy, layer)
C: LONG DeleteLayer(LONG, struct Layer *)
ML: d0 = DeleteLayer(a0, a1)
Arguments: dummy = unused
layer = the layer to delete
Result: success = zero if unsuccessful

DeleteMsgPort (Revision 2.0)

Description: frees a message port created by CreateMsgPort( )
Library: exec.library
Offset: -$2A0
Syntax: DeleteMsgPort(msgPort)
C: VOID DeleteMsgPort(struct MsgPort *)
ML: DeleteMsgPort(a0)
Arguments: msgPort = the message port to delete; may be 0
Result: none

DeletePool (Revision 3.0)

Description: drains an entire memory pool
Library: exec.library

Offset: -$2BE
Syntax: DeletePool(poolHeader)
C: VOID DeletePool(VOID *)
ML: DeletePool(a0)
Arguments: poolHeader = as returned by CreatePool( )
Result: none

DeleteRexxMsg (Revision 2.0)
Description: frees RexxMsg structure allocated by CreateRexxMsg( )
Library: rexxsyslib.library
Offset: -$96
Syntax: DeleteRexxMsg(rexxmsg)
C: VOID DeleteRexxMsg(struct RexxMsg *)
ML: DeleteRexxMsg(a0)
Arguments: packet = RexxMsg structure returned by CreateRexxMsg( )
Result: none

DeleteVar (Revision 2.0)
Description: deletes a local or environment variable
Library: dos.library
Offset: -$390
Syntax: success = DeleteVar(name, flags)
C: BOOL DeleteVar(STRPTR, ULONG)
ML: d0 = DeleteVar(d1, d2)
Arguments: name = variable name
flags = type of variable to delete—GVF_LOCAL_ONLY ($00000100), GVF_GLOBAL_ONLY ($00000200)
Result: success = zero if unsuccessful

DeviceProc
Description: returns the process MsgPort of a device
Library: dos.library
Offset: -$AE
Syntax: process = DeviceProc(name)
C: struct MsgPort *DeviceProc (STRPTR)
ML: d0 = DeviceProc(d1)
Arguments: name = name of device under investigation
Result: none

Disable
Description: disables interrupt processing
Library: exec.library
Offset: -$78
Syntax: Disable( );
C: VOID Disable(VOID)

ML: Disable( )
Arguments: none
Result: none

DisownBlitter

Description: frees the blitter so other tasks may use it
Library: graphics.library
Offset: -$1CE
Syntax: DisownBlitter( )
C: VOID DisownBlitter(VOID)
ML: DisownBlitter( )
Arguments: none
Result: none

DisplayAlert

Description: creates an alert
Library: intuition.library
Offset: -$5A
Syntax: response = DisplayAlert(alertNumber, string, height)
C: BOOL DisplayAlert(ULONG, UBYTE *, WORD)
ML: d0 = DisplayAlert(d0, a0, d1)
Arguments: alertNumber = bit flags specifying alert type—RECOVERY_ALERT ($00000000), DEADEND_ALERT ($80000000)
string = message to be displayed by alert
height = minimum raster lines required to display alert message
Result: response = nonzero if the left mouse button was pressed; zero if the right mouse button was pressed or if this is a DEADEND_ALERT

DisplayBeep

Description: flashes a screen's border
Library: intuition.library
Offset: -$60
Syntax: DisplayBeep(screen)
C: VOID DisplayBeep(struct Screen *)
ML: DisplayBeep(a0)
Arguments: screen = screen to be "beeped"; 0 to flash the border of every screen
Result: none

DisposeCxMsg (Revision 2.0)

Description: frees a commodity message
Library: commodities.library
Offset: -$A8
Syntax: DisposeCxMsg(cxm)
C: VOID DisposeCxMsg(struct CxMsg *)

ML: DisposeCxMsg(a0)
Arguments: cxm = commodity message to free
Result: none

DisposeDTObject (Revision 3.0)
Description: frees a DataType object
Library: datatypes.library
Offset: -$36
Syntax: DisposeDTObject(object)
C: VOID DisposeDTObject(Object *)
ML: DisposeDTObject(a0)
Arguments: object = Object structure returned by NewDTObjectA( )
Result: none

DisposeFontContents (Revision 2.0)
Description: frees the array of FontContents structures returned by NewFontContents( )
Library: diskfont.library
Offset: -$30
Syntax: DisposeFontContents(fontContentsHeader)
C: VOID DisposeFontContents(struct FontContentsHeader *)
ML: DisposeFontContents(a1)
Arguments: fontContentsHeader = FontContentsHeader structure returned by NewFontContents( )
Result: none

DisposeLayerInfo
Description: frees all the memory allocated for a LayerInfo
Library: layers.library
Offset: -$96
Syntax: DisposeLayerInfo(li)
C: VOID DisposeLayerInfo(struct Layer_Info *)
ML: DisposeLayerInfo(a0)
Arguments: li = the LayerInfo structure to free
Result: none

DisposeObject (Revision 2.0)
Description: deletes a boopsi object
Library: intuition.library
Offset: -$282
Syntax: DisposeObject(object)
C: VOID DisposeObject(APTR)
ML: DisposeObject(a0)
Arguments: object = unwanted boopsi object, as returned by NewObject( )
Result: none

DisposeRegion

Description: returns memory used by a Region structure
Library: graphics.library
Offset: -$216
Syntax: DisposeRegion(region)
C: VOID DisposeRegion(struct Region *)
ML: DisposeRegion(a0)
Arguments: region = Region structure to dispose of
Result: none

DivertCxMsg (Revision 2.0)

Description: sends a commodity message to an object list
Library: commodities.library
Offset: -$9C
Syntax: DivertCxMsg(cxm, headObj, returnObj)
C: VOID DivertCxMsg(struct CxMsg *, CxObj *, CxObj *)
ML: DivertCxMsg(a0, a1, a2)
Arguments: cxm = commodity message to divert
headObj = list of objects to receive the diverted message
returnObj = object to use as a place holder
Result: none

DoAsyncLayout (Revision 3.0)

Description: calls a DataType object's DTM_ASYNCLAYOUT method as a separate process
Library: datatypes.library
Offset: -$54
Syntax: retval = DoAsyncLayout(object, gpl)
C: ULONG DoAsyncLayout(Object *, struct gpLayout *)
ML: do = DoAsyncLayout(a0, a1)
Arguments: object = Object structure returned by NewDTObjectA( )
gpl = gpLayout message
Result: retval = zero if unsuccessful

DoCollision

Description: tests every gel in a gel list for collisions
Library: graphics.library
Offset: -$6C
Syntax: DoCollision(rastPort)
C: VOID DoCollision(struct RastPort *)
ML: DoCollision(a1)
Arguments: rastPort = RastPort with gel list
Result: none
The gel collision-handler routine (as defined by the SetCollision( ) function) is called if a collision is detected.

DoDTMethodA (Revision 3.0)

Description: executes a specific method of a DataType object
Library: datatypes.library
Offset: -$5A
Syntax: retval = DoDTMethodA(object, window, requester, message)
C: ULONG DoDTMethodA(Object *, struct Window *, struct Requester *, Msg)
ML: d0 = DoDTMethodA(a0, a1, a2, a3)
Arguments: object = Object structure returned by NewDTObjectA( )
window = window to which the object is attached
requester = requester to which the object is attached
message = message to send to the object
Result: retval = value returned by method

DoGadgetMethodA (Revision 3.0)

Description: invokes a method on a boopsi gadget
Library: intuition.library
Offset: -$32A
Syntax: result = DoGadgetMethodA(gadget, window, requester, message)
C: ULONG DoGadgetMethod(struct Gadget *, struct Window *, struct Requester *, Msg)
ML: d0 = DoGadgetMethodA(a0, a1, a2, a3)
Arguments: gadget = abstract pointer to a boopsi gadget
window = window containing boopsi the gadget
requester = requester for REQGADGETs
message = boopsi message to send
Result: result = result value as defined by boopsi gadget's method

DoHookClipRects (Revision 3.0)

Description: calls the appropriate hook for a group of ClipRects
Library: layers.library
Offset: -$D8
Syntax: DoHookClipRects(hook, rport, rect)
C: VOID DoHookClipRects(struct Hook *, struct RastPort *, struct Rectangle *)
ML: DoHookClipRects(a0, a1, a2)
Arguments: hook = the layer callback Hook to call
rport = the RastPort structure on which to operate
rect = bounding rectangle for operation
Result: none

DoIO

Description: executes an I/O command and waits for its completion
Library: exec.library
Offset: -$1C8

Syntax: error = DoIO(iORequest)
C: BYTE DoIO(struct IORequest *)
ML: d0 = DoIO(a1)
Arguments: iORequest = an IORequest initialized by OpenDevice( )
Result: error = a sign-extended copy of the io_Error field of the IORequest
Most device commands require that the error return be checked.

DoPkt (Revision 2.0)

Description: sends a DOS packet and waits for reply
Library: dos.library
Offset: -$F0
Syntax: result1 = DoPkt(port, action, arg1, arg2, arg3, arg4, arg5)
C: LONG DoPkt(struct MsgPort *, LONG, LONG, LONG, LONG, LONG, LONG)
ML: d0 = DoPkt(d1, d2, d3, d4, d5, d6, d7)
Arguments: port = pr_MsgPort of the handler process to receive packet
action = the action requested of the filesystem/handler arg1, arg2, arg3, arg4,arg5 = packet arguments; depend on the action requested
Result: result1 = the value returned in dp_Res1; zero if unsuccessful

DoubleClick

Description: compares two mouse clicks against a double-click
Library: intuition.library
Offset: -$66
Syntax: isDouble = DoubleClick(firstSecs, firstMicros, secondSeconds, secondMicros)
C: BOOL DoubleClick(ULONG, ULONG, ULONG, ULONG)
ML: d0 = DoubleClick(d0, d1, d2, d3)
Arguments: firstSeconds, firstMicros = the time at which the first mouse click occured
secondSeconds, secondMicros = the time at which the second mouse click occured
Result: isDouble = nonzero if the two mouse clicks should be considered one double-click

Draw

Description: draws a line from the current pen position to a specified point
Library: graphics.library
Offset: -$72
Syntax: Draw(rastPort, x, y)
C: VOID Draw(struct RastPort *, WORD, WORD)
ML: Draw(a1, d0, d1)
Arguments: rastPort = destination RastPort
x, y = line endpoint
Result: none

DrawBevelBoxA (Revision 2.0)

Description: draws a bevelled box
Library: gadtools.library
Offset: -$78

Syntax: DrawBevelBoxA(rport, left, top, width, height, tags)
C: VOID DrawBevelBox(struct RastPort *, WORD, WORD, WORD, WORD, struct TagItem *)
ML: DrawBevelBoxA(a0, d0, d1, d2, d3, a1)
Arguments: rport = RastPort into which to draw the box
left = left coordinate of the box
top = top coordinate of the box
width = width of the box
height = height of the box
tagList = tag list specifying additional attributes—GTBB_Recessed ($00000033), GTBB_FrameType ($0000004D), GT_VisualInfo ($00000034)
Result: none

DrawBorder

Description: draws a border
Library: intuition.library
Offset: -$6C
Syntax: DrawBorder(rastPort, border, leftOffset, topOffset)
C: VOID DrawBorder(struct RastPort *, struct Border *, WORD, WORD)
ML: DrawBorder(a0, a1, d0, d1)
Arguments: rastPort = RastPort in which to draw border
border = initialized Border structure
leftOffset = horizontal offset from left edge of the RastPort
topOffset = vertical offset from top edge of the RastPort
Result: none

DrawEllipse

Description: draws an outlined ellipse
Library: graphics.library
Offset: -$B4
Syntax: DrawEllipse(rastPort, cx, cy, width, height)
C: VOID DrawEllipse(struct RastPort *, WORD, WORD, WORD, WORD)
ML: DrawEllipse(a1, d0, d1, d2, d3)
Arguments: rastPort = RastPort structure
cx, cy = ellipse's centerpoint relative to the rastport
width = horizontal radius of the ellipse; must be greater than zero
height = vertical radius of the ellipse; must be greater than zero
Result: none

DrawGList

Description: processes a gel list, queueing VSprites and drawing Bobs
Library: graphics.library
Offset: -$72
Syntax: DrawGList(rastPort, viewPort)
C: VOID DrawGList(struct RastPort *, struct ViewPort *)

ML: DrawGList(a1, a0)
Arguments: rastPort = RastPort where Bobs will be drawn
viewPort = ViewPort for which VSprites will be created
Result: none

DrawImage

Description: draws the image defined by an Image structure
Library: intuition.library
Offset: -$72
Syntax: DrawImage(rastPort, image, leftOffset, topOffset)
C: VOID DrawImage(struct RastPort *, struct Image *, WORD, WORD)
ML: DrawImage(a0, a1, d0, d1)
Arguments: rastPort = RastPort in which to draw image
image = initialized Image structure describing image
leftOffset = horizontal offset from left edge of RastPort
topOffset = vertical offset from top edge of RastPort
Result: none

DrawImageState (Revision 2.0)

Description: draws an extended Image with special attributes
Library: intuition.library
Offset: -$26A
Syntax: DrawImageState(rastPort, image, leftOffset, topOffset, state, drawInfo)
C: VOID DrawImageState(struct RastPort *, struct Image *, WORD, WORD, ULONG, struct DrawInfo *)
ML: DrawImageState(a0, a1, d0, d1, d2, a2)
Arguments: rastPort = RastPort in which to draw image
image = initialized Image structure describing image
leftOffset, topOffset = offset from edge of RastPort
state = visual state of Image:
IDS_NORMAL ($00000000), IDS_SELECTED ($00000001), IDS_DISABLED ($00000002), IDS_BUSY ($00000003), IDS_INDETERMINATE ($00000004), IDS_INACTIVENORMAL ($00000005), IDS_INACTIVESELECTED ($00000006), IDS_INACTIVEDISABLED ($00000007)
drawInfo = DrawInfo structure with pen selections and resolution info
Result: none

DupLock

Description: duplicates a lock
Library: dos.library
Offset: -$60
Syntax: lock = DupLock(lock)
C: BPTR DupLock(BPTR)

ML: d0 = DupLock(d1)
Arguments: lock = lock to duplicate
Result: newLock = duplicate of specified lock

DupLockFromFH (Revision 2.0)
Description: returns the lock of a file specified by a filehandle
Library: dos.library
Offset: -$174
Syntax: lock = DupLockFromFH(fh)
C: BPTR DupLockFromFH(BPTR)
ML: d0 = DupLockFromFH(d1)
Arguments: fh = filehandle of an open file
Result: lock = lock of specified file; zero if unsuccessful

EasyRequestArgs (Revision 2.0)
Description: provides a Revision 2.0 alternative to AutoRequest( )
Library: intuition.library
Offset: -$24C
Syntax: num = EasyRequestArgs(window, easyStruct, idcmpPtr, argList)
C: LONG EasyRequestArgs(struct Window *, struct EasyStruct *, ULONG *, APTR)
ML: d0 = EasyRequestArgs(a0, a1, a2, a3)
Arguments: window = window on the screen to receive the requester;
0 to indicate the Workbench (or the current public screen)
easyStruct = initialized EasyStruct structure
idcmpPtr = the IDCMP flags for requester
argList = arguments for format commands
Result: num = gadget number; -1 if an IDCMP message caused requester to close

Enable
Description: permits system interrupts to resume
Library: exec.library
Offset: -$7E
Syntax: Enable( );
C: VOID Enable(VOID)
ML: Enable( )
Arguments: none
Result: none

EndNotify (Revision 2.0)
Description: ends a notification request
Library: dos.library
Offset: -$37E
Syntax: EndNotify(notifystructure)

C: VOID EndNotify(struct NotifyRequest *)
ML: EndNotify(d1)
Arguments: notifystructure = structure passed to StartNotify( )
Result: none

EndRefresh

Description: ends the optimized refresh state of a window
Library: intuition.library
Offset: -$16E
Syntax: EndRefresh(window, complete)
C: VOID EndRefresh(struct Window *, BOOL)
ML: EndRefresh(a0, d0)
Arguments: window = window in optimized-refresh mode
complete = nonzero if the window is completely refreshed
Result: none

EndRequest

Description: removes a currently active requester
Library: intuition.library
Offset: -$78
Syntax: EndRequest(requester, window)
C: VOID EndRequest(struct Requester *, struct Window *)
ML: EndRequest(a0, a1)
Arguments: requester = unwanted requester
window = window that owns the unwanted requester
Result: none

EndUpdate

Description: removes damage list and restores layer to normal state
Library: layers.library
Offset: -$54
Syntax: EndUpdate(layer, complete)
C: VOID EndUpdate(struct Layer *, UWORD)
ML: EndUpdate(a0, d0)
Arguments: layer = the layer
complete = nonzero if the update is complete; zero if the damage list should be retained
Result: none

Enqueue

Description: inserts or appends a node to a system queue
Library: exec.library
Offset: -$10E
Syntax: Enqueue(list, node)
C: VOID Enqueue(struct List *, struct Node *)

ML: Enqueue(a0, a1)
Arguments: list = the system queue header
node = the node to enqueue; this must be a full-featured node with type, priority, and name fields
Result: none

EnqueueCxObj (Revision 2.0)
Description: inserts a commodity object into a prioritized list of objects
Library: commodities.library
Offset: -$5A
Syntax: EnqueueCxObj(headObj, cxobj)
C: VOID EnqueueCxObj(CxObj *, CxObj *)
ML: EnqueueCxObj(a0, a1)
Arguments: headObj = list of objects
cxobj = commodity object to add
Result: none

EntryHandler (Revision 2.0)
Description: adds an entry handler to the IFFHandle context
Library: iffparse.library
Offset: -$66
Syntax: error = EntryHandler(iff, type, id, position, handler, object)
C: LONG EntryHandler(struct IFFHandle *, LONG, LONG, LONG, struct Hook *, APTR)
ML: d0 = EntryHandler(a0, d0, d1, d2, a1, a2)
Arguments: iff = an IFFHandle structure
type = type code for chunk to handle
id = ID code for chunk to handle
position = local context item position—IFFSLI_ROOT ($00000001), IFFSLI_TOP ($00000002), IFFSLI_PROP ($00000003)
handler = a Hook structure
object = user-defined pointer which is passed to handler in a2
Result: error = zero if successful; nonzero error-code if unsuccessful

EraseImage (Revision 2.0)
Description: erases an Image
Library: intuition.library
Offset: -$276
Syntax: EraseImage(rastPort, image, leftOffset, topOffset)
C: VOID EraseImage(struct RastPort *, struct Image *, WORD, WORD)
ML: EraseImage(a0, a1, d0, d1)
Arguments: rastPort = RastPort to erase image from
image = image to erase
leftOffset, rightOffset = offset from edge of RastPort
Result: none

EraseRect (Revision 2.0)

Description: draws a filled rectangle using the current BackFill hook
Library: graphics.library
Offset: -$32A
Syntax: EraseRect(rastPort, x1, y1, x2, y2)
C: VOID EraseRect(struct RastPort *, WORD, WORD, WORD, WORD)
ML: EraseRect(a1, d0, d1, d2, d3)
Arguments: rastPort = RastPort structure
x1, y1 = top left corner of rectangle
x2, y2 = bottom right corner of rectangle
Result: none

ErrorReport (Revision 2.0)

Description: displays a Retry/Cancel requester to handle errors
Library: dos.library
Offset: -$1E0
Syntax: status = ErrorReport(code, type, arg1, device)
C: BOOL ErrorReport(LONG, LONG, ULONG, struct MsgPort *)
ML: d0 = ErrorReport(d1, d2, d3, a0)
Arguments: code = error code to display:
ERROR_DISK_NOT_VALIDATED ($000000D5),
ERROR_DISK_WRITE_PROTECTED ($000000D6),
ERROR_DISK_FULL ($000000DD),
ERROR_DEVICE_NOT_MOUNTED ($000000DA),
ERROR_NOT_A_DOS_DISK ($000000E1), ERROR_NO_DISK
($000000E2), ABORT_DISK_ERROR ($00000128),
ABORT_BUSY ($00000120)
type = requester type:
see <dos/dosextens.h> for description
arg1 = argument (depends on requester type)
device = the handler task for which report is to be made;
only required for REPORT_LOCK and arg1 = NULL
Result: status = nonzero if Cancel is selected; zero if Retry is selected

ExAll (Revision 2.0)

Description: examines an entire directory
Library: dos.library
Offset: -$1B0
Syntax: continue = ExAll(lock, buffer, size, type, control)
C: BOOL ExAll(BPTR, STRPTR, LONG, LONG, struct ExAllControl *)
ML: d0 = ExAll(d1, d2, d3, d4, d5)

Arguments: lock = lock on directory to be examined
buffer = longword-aligned buffer to receive data
size = size of buffer
type = type of data to be returned
control = control data structure allocated by AllocDosObject
Result: continue = nonzero if there is more directory information pending; zero if entire directory has been examined

ExAllEnd (Revision 3.0)

Description: stops an ExAll( ) directory examination before it runs out of directory information (allowing DOS to handle filesystems that can't abort ExAll( ) directly)
Library: dos.library
Offset: -$3D8
Syntax: ExAllEnd(lock, buffer, size, type, control)
C: ExAllEnd(BPTR, STRPTR, LONG, LONG, struct ExAllControl *)
ML: ExAllEnd(d1, d2, d3, d4, d5)
Arguments: lock = lock on directory to be examined
buffer = longword-aligned buffer for data returned
size = size of buffer
type = type of data to be returned
control = control data structure allocated by AllocDosObject
Result: none

Examine

Description: examines a directory or file associated with a lock
Library: dos.library
Offset: -$66
Syntax: success = Examine(lock, FileInfoBlock)
C: BOOL Examine(BPTR, struct FileInfoBlock *)
ML: d0 = Examine(d1, d2)
Arguments: lock = lock of file or directory to be examined
infoBlock = FileInfoBlock that will hold file or directory information
Result: success = zero if unsuccessful

ExamineFH (Revision 2.0)

Description: gets information about an open file
Library: dos.library
Offset: -$186
Syntax: success = ExamineFH(fh, fib)
C: BOOL ExamineFH(BPTR, struct FileInfoBlock *)
ML: d0 = ExamineFH(d1, d2)
Arguments: fh = filehandle to examine
fib = FileInfoBlock to receive data
Result: success = zero if unsuccessful

Execute

Description: executes a CLI command
Library: dos.library
Offset: -$DE
Syntax: success = Execute(commandString, input, output)
C: BOOL Execute(STRPTR, BPTR, BPTR)
ML: d0 = Execute(d1, d2, d3)
Arguments: commandString = CLI command to execute
input = filehandle to be used for CLI command's input
output = filehandle to be used for CLI command's output
Result: success = zero if unsuccessful;
note that this return value is notoriously inaccurate

Exit

Description: exits from a program
Library: dos.library
Offset: -$90
Syntax: Exit(returnCode)
C: VOID Exit(LONG)
ML: Exit(d1)
Arguments: returnCode = AmigaDOS return code; zero for no error
Result: none

ExitHandler (Revision 2.0)

Description: adds an exit handler to the IFFHandle context
Library: iffparse.library
Offset: -$6C
Syntax: error = ExitHandler(iff, type, id, position, handler, object)
C: LONG ExitHandler(struct IFFHandle *, LONG, LONG, LONG, struct Hook *, APTR)
ML: d0 = ExitHandler(a0, d0, d1, d2, a1, a2)
Arguments: iff = pointer to IFFHandle structure
type = type code for chunk to handle
id = identifier code for chunk to handle
position = local context item position:
IFFSLI_ROOT ($00000001), IFFSLI_TOP ($00000002),
IFFSLI_PROP ($00000003)
handler = pointer to Hook structure
object = user-defined pointer which is passed to handler in a2
Result: error = zero if successful; nonzero error-code if unsuccessful

ExNext

Description: examines the next entry in a directory
Library: dos.library
Offset: -$6C
Syntax: success = ExNext(lock, FileInfoBlock)
C: BOOL ExNext(BPTR, struct FileInfoBlock *)
ML: d0 = ExNext(d1, d2)
Arguments: lock = lock returned by Examine( )
infoBlock = info block to receive entry information
Result: success = zero if unsuccessful

ExtendFont (Revision 2.0)

Description: guarantees tf_Extension has been built for a font
Library: graphics.library
Offset: -$330
Syntax: success = ExtendFont(font, tags)
C: ULONG ExtendFontTags(struct TextFont *, struct TagItem *)
ML: d0 = ExtendFont(a0, a1)
Arguments: font = font to extend
tags = tag list of attributes—TA_DeviceDPI ($80000001); 0 for default font
Result: success = zero if unsuccessful

FattenLayerInfo (Obsolete under Revision 1.2)

Description: converts a 1.0 LayerInfo to a 1.1 LayerInfo
Library: layers.library
Offset: -$9C
Syntax: FattenLayerInfo(li)
C: LONG FattenLayerInfo(struct Layer_Info *)
ML: FattenLayerInfo(a0)
Arguments: li = the LayerInfo structure to convert
Result: none

Fault (Revision 2.0)

Description: returns the text associated with a DOS error code
Library: dos.library
Offset: -$1D4
Syntax: success = Fault(code, header, buffer, len)
C: BOOL Fault(LONG, STRPTR, STRPTR, LONG)
ML: d0 = Fault(d1, d2, d3, d4)
Arguments: code = error code
header = text to output before error text
buffer = buffer to receive error message
len = size of buffer
Result: success = zero if unsuccessful

FGetC (Revision 2.0)

Description: reads a character from the specified filehandle
Library: dos.library
Offset: -$132
Syntax: char = FGetC(fh)
C: LONG FGetC(BPTR)
ML: d0 = FGetC(d1)
Arguments: fh = filehandle to use for buffered input
Result: char = character read

FGets (Revision 2.0)

Description: reads a line from the specified filehandle
Library: dos.library
Offset: -$150
Syntax: buffer = FGets(fh, buf, len)
C: STRPTR FGets(BPTR, STRPTR, ULONG)
ML: d0 = FGets(d1, d2, d3)
Arguments: fh = filehandle to use for buffered input
buf = buffer to receive data
len = size of buffer
Result: pointer to buffer; zero if unsuccessful

FilePart (Revision 2.0)

Description: returns the last component of a pathname
Library: dos.library
Offset: -$366
Syntax: fileptr = FilePart(path)
C: STRPTR FilePart(STRPTR)
ML: d0 = FilePart(d1)
Arguments: path = pathname to investigate
Result: fileptr = pointer to the last component of the pathname

FillRexxMsg (Revision 2.0)

Description: installs argument strings in a RexxMsg structure
Library: rexxsyslib.library
Offset: -$A2
Syntax: success = FillRexxMsg(msgptr, count, mask)
C: BOOL FillRexxMsg(struct RexxMsg *,ULONG,ULONG)
ML: d0 = FillRexxMsg(a0, d0, d1)
Arguments: msgptr = RexxMsg structure returned by CreateRexxMsg( )
count = number of argument slots to initialize; 1-16
mask = bit set to determine type of fields; clear bit indicates string pointer, set bit indicates integer
Result: success = zero if unsuccessful

FilterTagChanges (Revision 2.0)

Description: eliminates tags which specify no change
Library: utility.library
Offset: -$36
Syntax: FilterTagChanges(changeList, originalList, apply)
C: VOID FilterTagChanges(struct TagItem *, struct TagItem *, ULONG)
ML: FilterTagChanges(a0, a1, d0)
Arguments: changeList = tag list to use as delta
originalList = tag list to modify
apply = nonzero if originalList's data values should be updated
Result: none

FilterTagItems (Revision 2.0)

Description: removes selected items from a tag list
Library: utility.library
Offset: -$60
Syntax: numValid = FilterTagItems(tags, filterArray, logic)
C: ULONG FilterTagItems(struct TagItem *, Tag *, ULONG)
ML: d0 = FilterTagItems(a0, a1, d0)
Arguments: tags = tag list to be filtered
filterArray = array of tag values terminated by TAG_DONE
logic = flag to indicate inclusion or exclusion of tags specified in filterArray:
TAGFILTER_AND ($00000000), TAGFILTER_NOT ($00000001)
Result: numValid = number of items left in the filtered list

FindArg (Revision 2.0)

Description: finds a keyword in a template
Library: dos.library
Offset: -$324
Syntax: index = FindArg(template, keyword)
C: LONG FindArg(STRPTR, STRPTR)
ML: d0 = FindArg(d1, d2)
Arguments: keyword = keyword to search for in template
template = template string to search
Result: index = number of entry in template; -1 if not found

FindCliProc (Revision 2.0)

Description: returns a pointer to the requested CLI process
Library: dos.library
Offset: -$222
Syntax: proc = FindCliProc(num)
C: struct Process *FindCliProc(LONG)
ML: d0 = FindCliProc(d1)
Arguments: num = task number of CLI process
Result: proc = the CLI process; zero if not found

FindCollection (Revision 2.0)

Description: obtains a pointer to the current list of collection items
Library: iffparse.library
Offset: -$A2
Syntax: ci = FindCollection(iff, type, id)
C: struct CollectionItem *FindCollection(struct IFFHandle *, LONG, LONG)
ML: d0 = FindCollection(a0, d0, d1)
Arguments: iff = an IFFHandle structure
type = type code to find
id = id code to find
Result: ci = the last collection chunk encountered matching the specifications;
zero if no chunk was found

FindColor (Revision 3.0)

Description: finds the closest matching color in a colormap
Library: graphics.library
Offset: -$3F0
Syntax: color = FindColor(colorMap, red, green, blue, maxPen)
C: ULONG FindColor(struct ColorMap *, ULONG, ULONG, ULONG, LONG)
ML: d0 = FindColor(a3, d1, d2, d3, d4)
Arguments: colorMap = ColorMap structure
red, green, blue = color attributes
maxPen = maximum color number; -1 for all available colors
Result: color = best matching color

FindConfigDev

Description: finds a matching ConfigDev entry
Library: expansion.library
Offset: -$48
Syntax: configDev = FindConfigDev(oldConfigDev, manufacturer, product)
C: struct ConfigDev *FindConfigDev(struct ConfigDev *, LONG, LONG)
ML: d0 = FindConfigDev(a0, d0, d1)
Arguments: oldConfigDev = element in list of ConfigDev structures after which
to start searching;
0 to start at beginning
manufacturer = manufacturer code to search for;
-1 to ignore manufacturer codes
product = product code to search for; -1 to ignore product codes
Result: configDev = the first ConfigDev entry found matching the specification;
zero if no match was found

FindDisplayInfo (Revision 2.0)

Description: searches for a record identified by a key
Library: graphics.library
Offset: -$2D6

Syntax: handle = FindDisplayInfo(id)
C: DisplayInfoHandle FindDisplayInfo(ULONG)
ML: d0 = FindDisplayInfo(d0)
Arguments: id = key identifier
Result: handle = Record handle; zero if unsuccessful

FindDosEntry (Revision 2.0)

Description: finds a DosList entry in a locked device list
Library: dos.library
Offset: -$2AC
Syntax: newdlist = FindDosEntry(dlist, name, flags)
C: struct DosList *FindDosEntry(struct DosList *, STRPTR, ULONG)
ML: d0 = FindDosEntry(d1, d2, d3)
Arguments: dlist = device entry from which to begin the search
name = name of device entry (without :) to locate
flags = search control flags:
LDF_DEVICES ($00000004), LDF_VOLUMES ($00000008), LDF_ASSIGNS ($00000010), LDF_ENTRY ($00000020);
use only the flags you used when locked the DosList
Result: newdlist = device entry; zero if unsuccessful

FindLocalItem (Revision 2.0)

Description: gets a local context item from the context stack
Library: iffparse.library
Offset: -$D2
Syntax: lci = FindLocalItem(iff, type, id, ident)
C: struct LocalContextItem *FindLocalItem(struct IFFHandle *, LONG, LONG, LONG)
ML: d0 = FindLocalItem(a0, d0, d1, d2)
Arguments: iff = an IFFHandle structure
type = type code to find
id = id code to find
ident = ident code for the class of context item to find
Result: lci = a local context item; zero if no item was found

FindName

Description: finds a system list node by name
Library: exec.library
Offset: -$114
Syntax: node = FindName(start, name)
C: struct Node *FindName(struct List *, STRPTR)
ML: d0 = FindName(a0, a1)

Arguments: start = a list header or a list node to start the search; if a node, this one is skipped
name = name of the node to find
Result: node = the node matching the name given; zero if unsuccessfull

FindNamedObject (Revision 3.0)
Description: finds the next object with a given name and increments its usage count
Library: utility.library
Offset: -$F0
Syntax: object = FindNamedObject(namespace, name, lastobject)
C: struct NamedObject *FindNamedObject(struct NamedObject *, char *, struct NamedObject *)
ML: d0 = FindNamedObject(a0, a1, a2)
Arguments: namespace = the namespace to search
name = the name of the object to find; 0 indicates all objects
lastobject = the starting point for the search; 0 indicates the beginning
Result: object = the first match found; zero if no object found

FindPort
Description: finds a system message port by name
Library: exec.library
Offset: -$186
Syntax: port = FindPort(name)
C: struct MsgPort *FindPort(STRPTR)
ML: d0 = FindPort(a1)
Arguments: name = name of the port to find
Result: port = the message port matching the name; zero if unsuccessfull

FindProp (Revision 2.0)
Description: gets a stored property chunk
Library: iffparse.library
Offset: -$9C
Syntax: sp = FindProp(iff, type, id)
C: struct StoredProperty *FindProp(struct IFFHandle *, LONG, LONG)
ML: d0 = FindProp(a0, d0, d1)
Arguments: iff = an IFFHandle structure
type = type code to find
id = id code to find
Result: sp = a stored property chunk; zero if no chunk was found

FindPropContext (Revision 2.0)
Description: gets the property context for the current state
Library: iffparse.library

Offset: -$A8
Syntax: cn = FindPropContext(iff)
C: struct ContextNode *FindPropContext(struct IFFHandle *)
ML: d0 = FindPropContext(a0)
Arguments: iff = an IFFHandle structure
Result: cn = ContextNode of property scoping chunk

FindResident

Description: finds a resident module by name
Library: exec.library
Offset: -$60
Syntax: resident = FindResident(name)
C: struct Resident *FindResident(STRPTR)
ML: d0 = FindResident(a1)
Arguments: name = name of the module to find
Result: resident = the resident tag structure matching the name; zero if unsuccessfull

FindSegment (Revision 2.0)

Description: finds a segment on the resident list
Library: dos.library
Offset: -$30C
Syntax: segment = FindSegment(name, start, type)
C: struct Segment *FindSegment(STRPTR, struct Segment *, LONG)
ML: d0 = FindSegment(d1, d2, d3)
Arguments: name = name of segment to find
start = segment from which to begin the search
type = which segment to find; zero for normal segment, nonzero for system segment
Result: segment = the segment found; zero if unsuccessful

FindSemaphore

Description: finds a system signal semaphore by name
Library: exec.library
Offset: -$252
Syntax: signalSemaphore = FindSemaphore(name)
C: struct SignalSemaphore *FindSemaphore(STRPTR)
ML: d0 = FindSemaphore(a1)
Arguments: name = name of the semaphore to find
Result: semaphore = the signal semaphore matching the name; zero if unsuccessful

FindTagItem (Revision 2.0)

Description: scans a tag list for a specific tag
Library: utility.library

Offset: -$1E
Syntax: tag = FindTagItem(tagValue, tags)
C: struct TagItem *FindTagItem(Tag, struct TagItem *)
ML: d0 = FindTagItem(d0, a0)
Arguments: tagValue = tag value to find
tags = tag list to search
Result: tag = the item with ti_Tag matching tagValue; zero if no match was found

FindTask

Description: finds a task by name, or finds oneself
Library: exec.library
Offset: -$126
Syntax: task = FindTask(name)
C: struct Task *FindTask(STRPTR)
ML: d0 = FindTask(a1)
Arguments: name = name of the task to find; 0 to find oneself
Result: task = the task (or process) matching the name; zero if unsuccessfull

FindToolType

Description: retrieves the value of a ToolType variable
Library: icon.library
Offset: -$60
Syntax: value = FindToolType(toolTypeArray, typeName)
C: char *FindToolType(char **, char *)
ML: d0 = FindToolType(a0, a1)
Arguments: toolTypeArray = an array of ToolType variable strings
typeName = the name of the ToolType entry to find
Result: value = the string bound to typeName in toolTypeArray; zero if typeName is not found

FindVar (Revision 2.0)

Description: finds a local variable
Library: dos.library
Offset: -$396
Syntax: var = FindVar(name, type)
C: struct LocalVar *FindVar(STRPTR, ULONG)
ML: d0 = FindVar(d1, d2)
Arguments: name = name of variable to find
Result: var = a LocalVar structure; zero if unsuccessful

Flood

Description: fills an area within a RastPort
Library: graphics.library
Offset: -$14A

Syntax: Flood(rastPort, mode, x, y)
C: Flood(struct RastPort *, ULONG, WORD, WORD)
ML: Flood(a1, d2, d0, d1)
Arguments: rastPort = RastPort to flood
x, y = coordinate at which to start flooding
mode = 0 to fill pixels that are the same color as AOLPen, 1 to fill pixels that are the same color as the pixel specified by the x and y arguments
Result: none

Flush (Revision 2.0)
Description: flushes buffers for a buffered filehandle
Library: dos.library
Offset: -$168
Syntax: success = Flush(fh)
C: LONG Flush(BPTR)
ML: d0 = Flush(d1)
Arguments: fh = filehandle to flush
Result: success = zero if unsuccessful

FontExtent (Revision 2.0)
Description: returns the font attributes of the current font
Library: graphics.library
Offset: -$2FA
Syntax: FontExtent(font, fontExtent)
C: VOID FontExtent(struct TextFont *, struct TextExtent *)
ML: FontExtent(a0, a1)
Arguments: font = TextFont from which the font metrics are extracted
fontExtent = TextExtent structure to receive data
Result: none

Forbid
Description: disables task rescheduling
Library: exec.library
Offset: -$84
Syntax: Forbid( )
C: VOID Forbid(VOID)
ML: Forbid( )
Arguments: none
Result: none

Format (Revision 2.0)
Description: formats a disk (or any AmigaDOS filesystem media)
Library: dos.library
Offset: -$2CA
Syntax: success = Format(filesystem, volumename, dostype)

C: BOOL Format(STRPTR, STRPTR, ULONG)
ML: d0 = Format(d1, d2, d3)
Arguments: filesystem = name of device to be formatted, such as DF0:
volumename = name for volume (without :)
dostype = type of format, if filesystem supports multiple types
Result: success = zero if unsuccessful

FormatDate (Revision 3.0)

Description: generates a date string based on a locale
Library: locale.library
Offset: -$3C
Syntax: FormatDate(locale, string, date, putCharFunc)
C: VOID FormatDate(struct Locale *, STRPTR, struct DateStamp *, struct Hook *)
ML: FormatDate(a0, a1, a2, a3)
Arguments: locale = the locale to use for the formatting
string = a template describing the output format
date = the date to format
putCharFunc = a callback function to process each character as it is generated
Result: none

FormatString (Revision 3.0)

Description: formats data into a character stream
Library: locale.library
Offset: -$42
Syntax: next = FormatString(locale, string, dataStream, putCharFunc)
C: APTR FormatString(struct Locale *, STRPTR, APTR, struct Hook *)
ML: d0 = FormatString(a0, a1, a2, a3)
Arguments: locale = the locale to use for the formatting
string = a template describing the output format
dataStream = data to be formatted
putCharFunc = a callback function to process each character as it is generated
Result: next = pointer to just past the last data element used from dataStream

FPutC (Revision 2.0)

Description: writes a character to the specified filehandle
Library: dos.library
Offset: -$138
Syntax: char = FPutC(fh, char)
C: LONG FPutC(BPTR, LONG)
ML: d0 = FPutC(d1, d2)
Arguments: fh = filehandle to use for buffered output
char = character to write
Result: char = the character written if successful; EOF if unsuccessful

FPuts (Revision 2.0)

Description: writes a string to the specified filehandle
Library: dos.library
Offset: -$156
Syntax: error = FPuts(fh, str)
C: LONG FPuts(BPTR, STRPTR)
ML: d0 = FPuts(d1, d2)
Arguments: fh = filehandle to use for buffered output
str = string to write
Result: error = -1 if unsuccessful

FRead (Revision 2.0)

Description: reads multiple blocks from a filehandle
Library: dos.library
Offset: -$144
Syntax: count = FRead(fh, buf, blocklen, blocks)
C: LONG FRead(BPTR, STRPTR, ULONG, ULONG)
ML: d0 = FRead(d1, d2, d3, d4)
Arguments: fh = filehandle to use for buffered input
buf = buffer to receive data
blocklen = number of bytes per block
blocks = number of blocks to read
Result: count = number of blocks read; zero if completely unsuccessful

FreeArgs (Revision 2.0)

Description: frees RDArgs structure returned by ReadArgs( )
Library: dos.library
Offset: -$35A
Syntax: FreeArgs(rdargs)
C: VOID FreeArgs(struct RDArgs *)
ML: FreeArgs(d1)
Arguments: rdargs = structure returned from ReadArgs( )
Result: none

FreeAslRequest (Revision 2.0)

Description: frees a requester returned by AllocAslRequest( )
Library: asl.library
Offset: -$36
Syntax: FreeAslRequest(requester)
C: VOID FreeAslRequest(APTR)
ML: FreeAslRequest(a0)
Arguments: requester = requester returned by AllocAslRequest( )
Result: none

FreeBitMap (Revision 3.0)
Description: frees a bitmap created by the AllocBitMap( ) function
Library: graphics.library
Offset: -$39C
Syntax: FreeBitMap(bitMap)
C: VOID FreeBitMap(struct BitMap *)
ML: FreeBitMap(a0)
Arguments: bitMap = unwanted BitMap structure
Result: none

FreeClass (Revision 2.0)
Description: frees a boopsi class created by MakeClass( )
Library: intuition.library
Offset: -$2CA
Syntax: success = FreeClass(classPtr)
C: BOOL FreeClass(struct IClass *)
ML: d0 = FreeClass(a0)
Arguments: classPtr = class created by MakeClass( ).
Result: success = zero if unsuccessful

FreeColorMap
Description: frees a colormap created by the GetColorMap( ) function
Library: graphics.library
Offset: -$240
Syntax: FreeColorMap(colorMap)
C: VOID FreeColorMap(struct ColorMap *)
ML: FreeColorMap(a0)
Arguments: colorMap = unwanted colormap
Result: none

FreeConfigDev
Description: frees a ConfigDev structure returned by AllocConfigDev( )
Library: expansion.library
Offset: -$54
Syntax: FreeConfigDev(configDev)
C: VOID FreeConfigDev(struct ConfigDev *)
ML: FreeConfigDev(a0)
Arguments: configDev = ConfigDev structure returned by AllocConfigDev( )
Result: none

FreeCopList
Description: deallocates an intermediate copper list
Library: graphics.library
Offset: -$222
Syntax: FreeCopList(coplist)

C: VOID FreeCopList(struct CopList *)
ML: FreeCopList(a0)
Arguments: coplist = unwanted CopList structure
Result: none

FreeCprList
Description: deallocates a hardware copper list
Library: graphics.library
Offset: -$234
Syntax: FreeCprList(cprlist)
C: VOID FreeCprList(struct cprlist *)
ML: FreeCprList(a0)
Arguments: cprlist = unwanted cprlist structure
Result: none

FreeDBufInfo (Revision 3.0)
Description: frees a DBufInfo structure created by the AllocDBufInfo( ) function
Library: graphics.library
Offset: -$3CC
Syntax: FreeDBufInfo(db)
C: VOID FreeDBufInfo(struct DBufInfo *)
ML: FreeDBufInfo(a1)
Arguments: db = unwanted DBufInfo structure
Result: none

FreeDeviceProc (Revision 2.0)
Description: releases port returned by GetDeviceProc( )
Library: dos.library
Offset: -$288
Syntax: FreeDeviceProc(devproc)
C: VOID FreeDeviceProc(struct DevProc *)
ML: FreeDeviceProc(d1)
Arguments: devproc = value returned by GetDeviceProc( )
Result: none

FreeDiskObject
Description: frees all memory related to a Workbench disk object
Library: icon.library
Offset: -$5A
Syntax: FreeDiskObject(diskobj)
C: VOID FreeDiskObject(struct DiskObject *)
ML: FreeDiskObject(a0)
Arguments: diskobj = the DiskObject structure to free
Result: none

FreeDosEntry (Revision 2.0)

Description: frees an entry created by MakeDosEntry
Library: dos.library
Offset: -$2BE
Syntax: FreeDosEntry(dlist)
C: VOID FreeDosEntry(struct DosList *)
ML: FreeDosEntry(d1)
Arguments: dlist = DosList to free
Result: none

FreeDosObject (Revision 2.0)

Description: frees an object allocated by AllocDosObject( )
Library: dos.library
Offset: -$EA
Syntax: FreeDosObject(type, ptr)
C: VOID FreeDosObject(ULONG, VOID *)
ML: FreeDosObject(d1, d2)
Arguments: type = type passed to AllocDosObject( )
ptr = ptr returned by AllocDosObject( )
Result: none

FreeEntry

Description: frees multiple memory regions
Library: exec.library
Offset: -$E4
Syntax: FreeEntry(memList)
C: VOID FreeEntry(struct MemList *)
ML: FreeEntry(a0)
Arguments: memList = MemList structure initialized with MemEntry structures
Result: none

FreeExpansionMem

Description: frees expansion memory
Library: expansion.library
Offset: -$5A
Syntax: FreeExpansionMem(startSlot, numSlots)
C: VOID FreeExpansionMem(ULONG, ULONG)
ML: FreeExpansionMem(d0, d1)
Arguments: startSlot = the slot number allocated by AllocExpansionMem( )
numSlots = the number of slots to be freed
Result: none

FreeFileRequest (Revision 2.0—Obsolete under Revision 3.0)

Description: frees a FileRequester structure returned by AllocFileRequest( )
Library: asl.library
Offset: -$24
Syntax: FreeFileRequest(requester)
C: VOID FreeFileRequest(struct FileRequester *)
ML: FreeFileRequest(a0)
Arguments: requester = FileRequester structure returned by AllocFileRequest( )
Result: none

FreeFreeList

Description: frees all memory in a free list
Library: icon.library
Offset: -$36
Syntax: FreeFreeList(free)
C: VOID FreeFreeList(struct FreeList *)
ML: FreeFreeList(a0)
Arguments: free = a FreeList structure
Result: none

FreeGadgets (Revision 2.0)

Description: frees a list of gadgets
Library: gadtools.library
Offset: -$24
Syntax: FreeGadgets(glist)
C: VOID FreeGadgets(struct Gadget *)
ML: FreeGadgets(a0)
Arguments: glist = the first gadget in the list to free
Result: none

FreeGBuffers

Description: frees memory obtained by the GetGBuffers( ) function
Library: graphics.library
Offset: -$258
Syntax: FreeGBuffers(anOb, rastPort, db)
C: VOID FreeGBuffers(struct AnimOb *, struct RastPort *, BOOL)
ML: FreeGBuffers(a0, a1, d0)
Arguments: anOb = AnimOb structure
rastPort = current RastPort
db = nonzero if GBuffers are related to a double-buffered display
Result: none

FreeIFF (Revision 2.0)

Description: deallocates an IFFHandle structure
Library: iffparse.library
Offset: -$36
Syntax: FreeIFF(iff)
C: VOID FreeIFF(struct IFFHandle *)
ML: FreeIFF(a0)
Arguments: iff = an IFFHandle structure returned by AllocIFF( )
Result: none

FreeLocalItem (Revision 2.0)

Description: deallocates a local context item structure
Library: iffparse.library
Offset: -$CC
Syntax: FreeLocalItem(localItem)
C: VOID FreeLocalItem(struct LocalContextItem *)
ML: FreeLocalItem(a0)
Arguments: localItem = a LocalContextItem structure returned by
AllocLocalItem( )
Result: none

FreeMem

Description: deallocates memory
Library: exec.library
Offset: -$D2
Syntax: FreeMem(memoryBlock, byteSize)
C: VOID FreeMem(VOID *,ULONG)
ML: FreeMem(a1, d0)
Arguments: memoryBlock = the memory block to free
byteSize = the size of the desired block in bytes;
this will be rounded to a multiple of the system memory chunk size
Result: none

FreeMenus (Revision 2.0)

Description: frees memory allocated by CreateMenusA( )
Library: gadtools.library
Offset: -$36
Syntax: FreeMenus(menu)
C: VOID FreeMenus(struct Menu *)
ML: FreeMenus(a0)
Arguments: menu = a menu structure returned by CreateMenusA( )
Result: none

FreeNamedObject (Revision 3.0)

Description: frees a utility library object
Library: utility.library
Offset: -$F6
Syntax: FreeNamedObject(object)
C: VOID FreeNamedObject(VOID *)
ML: FreeNamedObject(a0)
Arguments: object = an object returned by AllocateNamedObjectA( )
Result: none

FreePooled (Revision 3.0)

Description: frees pooled memory
Library: exec.library
Offset: -$2CA
Syntax: FreePooled(poolHeader, memory, memSize)
C: VOID FreePooled(VOID *, VOID *, ULONG)
ML: FreePooled(a0, a1, d0)
Arguments: memory = the memory pool to free
poolHeader = a specific private pool header
Result: none

FreeRaster

Description: frees the bitplane returned by the AllocRaster( ) function
Library: graphics.library
Offset: -$1F2
Syntax: FreeRaster(bitplane, width, height)
C: VOID FreeRaster(PLANEPTR, UWORD, UWORD)
ML: FreeRaster(a0, d0, d1)
Arguments: bitplane = unwanted bitplane
width, height = dimensions of bitplane
Result: none

FreeRemember

Description: frees memory allocated with AllocRemember( ).
Library: intuition.library
Offset: -$198
Syntax: FreeRemember(rememberKey, reallyForget)
C: VOID FreeRemember(struct Remember **, BOOL)
ML: FreeRemember(a0, d0)
Arguments: rememberKey = address of a pointer to the Remember structure used by AllocRemember( )
reallyForget = zero to free the Remember nodes only; nonzero to free both the nodes and the memory buffers they reference
Result: none

FreeScreenBuffer (Revision 3.0)

Description: frees a ScreenBuffer structure
Library: intuition.library
Offset: -$306
Syntax: FreeScreenBuffer(screen, screenBuffer)
C: VOID FreeScreenBuffer(struct Screen *, struct ScreenBuffer *)
ML: FreeScreenBuffer(a0, a1)
Arguments: screen = screen containing unwanted ScreenBuffer
screenBuffer = ScreenBuffer to free
Result: none

FreeScreenDrawInfo (Revision 2.0)

Description: frees a DrawInfo structure obtained from GetScreenDrawInfo( )
Library: intuition.library
Offset: -$2B8
Syntax: FreeScreenDrawInfo(screen, drInfo)
C: VOID FreeScreenDrawInfo(struct Screen *, struct DrawInfo *)
ML: FreeScreenDrawInfo(a0, a1)
Arguments: screen = screen passed to GetScreenDrawInfo( )
drInfo = DrawInfo structure returned by GetScreenDrawInfo( )
Result: none

FreeSignal

Description: frees a signal bit
Library: exec.library
Offset: -$150
Syntax: FreeSignal(signalNum)
C: VOID FreeSignal(BYTE)
ML: FreeSignal(d0)
Arguments: signalNum = the signal number to free; 0-31
Result: none

FreeSprite

Description: frees the sprite allocated via the GetSprite( ) function
Library: graphics.library
Offset: -$19E
Syntax: FreeSprite(pick)
C: VOID FreeSprite(WORD)
ML: FreeSprite(d0)
Arguments: pick = sprite number as returned by the GetSprite( ) function
Result: none

FreeSpriteData (Revision 3.0)
Description: frees sprite data allocated by AllocSpriteData( ) function
Library: graphics.library
Offset: -$408
Syntax: FreeSpriteData(extsp)
C: VOID FreeSpriteData(struct ExtSprite *)
ML: FreeSpriteData(a2)
Arguments: extsp = the extended sprite structure to free; must not be 0
Result: none

FreeSysRequest
Description: frees resources allocated by BuildSysRequest( )
Library: intuition.library
Offset: -$174
Syntax: FreeSysRequest(window)
C: VOID FreeSysRequest(struct Window *)
ML: FreeSysRequest(a0)
Arguments: window = window pointer returned BuildSysRequest( )
Result: none

FreeTagItems (Revision 2.0)
Description: frees an allocated tag list
Library: utility.library
Offset: -$4E
Syntax: FreeTagItems(tags)
C: VOID FreeTagItems(struct TagItem *)
ML: FreeTagItems(a0)
Arguments: tags = tag list returned by AllocateTagItems( ), CloneTagItems( ), or MergeTagItems( )
Result: none

FreeTrap
Description: frees a processor trap
Library: exec.library
Offset: -$15C
Syntax: FreeTrap(trapNum)
C: VOID FreeTrap(ULONG)
ML: FreeTrap(d0)
Arguments: trapNum = the trap number to free; 0-15
Result: none

FreeVec (Revision 2.0)
Description: returns AllocVec( ) memory to the system
Library: exec.library
Offset: -$2B2

Syntax: FreeVec(memoryBlock)
C: VOID FreeVec(VOID *)
ML: FreeVec(a1)
Arguments: memoryBlock = the memory block to free; may be 0
Result: none

FreeVisualInfo (Revision 2.0)

Description: frees resources obtained by GetVisualInfo( )
Library: gadtools.library
Offset: -$84
Syntax: FreeVisualInfo(vi)
C: VOID FreeVisualInfo(APTR)
ML: FreeVisualInfo(a0)
Arguments: vi = a pointer returned by GetVisualInfoA( )
Result: none

FreeVPortCopLists

Description: deallocates all of a ViewPort's intermediate copper lists and their headers
Library: graphics.library
Offset: -$21C
Syntax: FreeVPortCopLists(viewPort)
C: VOID FreeVPortCopLists(struct ViewPort *)
ML: FreeVPortCopLists(a0)
Arguments: viewPort = ViewPort structure with unwanted copper lists
Result: none

FWrite (Revision 2.0)

Description: writes multiple blocks to a filehandle
Library: dos.library
Offset: -$14A
Syntax: count = FWrite(fh, buf, blocklen, blocks)
C: LONG FWrite(BPTR, STRPTR, ULONG, ULONG)
ML: d0 = FWrite(d1, d2, d3, d4)
Arguments: fh = filehandle to use for buffered output
buf = buffer containing data to be written
blocklen = number of bytes per block
blocks = number of blocks to read
Result: count = number of blocks written; zero if completely unsuccessful

GadgetMouse (Revision 2.0)

Description: calculates the gadget-relative mouse position
Library: intuition.library
Offset: -$23A
Syntax: GadgetMouse(gadget, gInfo, mousePoint)
C: VOID GadgetMouse(struct GadgetInfo *, WORD *)

ML: GadgetMouse(a0, a1, a2)
Arguments: gadget = gadget to which mouse is relative
gInfo = GadgetInfo structure passed to custom gadget hook routine
mousePoint = two words (a Point structure) to receive mouse coordinates
Result: none

GetAmigaGuideAttr (Revision 3.0)
Description: obtains attributes
Library: amigaguide.library
Offset -$72
Syntax: retval = GetAmigaGuideAttr(type, handle, buffer)
C: LONG GetAmigaGuideAttr(Tag, AMIGAGUIDECONTEXT, ULONG *)
ML: d0 = GetAmigaGuideAttr(d0, a0, a1)
Arguments: type = attribute to retrieve—AGA_Path ($80000001), AGA_XRefList ($80000002)
handle = handle to an AmigaGuide system
buffer = memory location to store data
Result: retval = zero if unsuccessful

GetAmigaGuideMsg (Revision 3.0)
Description: retrieves a message if available
Library: amigaguide.library
Offset: -$4E
Syntax: msg = GetAmigaGuideMsg(handle)
C: struct AmigaGuideMsg *GetAmigaGuideMsg(AMIGAGUIDECONTEXT)
ML: d0 = GetAmigaGuideMsg(a0)
Arguments: handle = handle to an AmigaGuide system
Result: msg = pointer to a message; zero if none available

GetAmigaGuideString (Revision 3.0)
Description: gets an AmigaGuide string
Library: amigaguide.library
Offset: -$D2
Syntax: txt = GetAmigaGuideString(id)
C: STRPTR GetAmigaGuideString(ULONG)
ML: d0 = GetAmigaGuidString(d0)
Arguments: id = valid string ID number
Result: txt = pointer to the string; zero if invalid ID

GetAPen (Revision 3.0)
Description: returns a RastPort's APen value
Library: graphics.library
Offset: -$35A
Syntax: pen = GetAPen(rastPort)

C: ULONG GetAPen(struct RastPort *)
ML: d0 = GetAPen (a0)
Arguments: rastPort = RastPort structure in question
Result: none

GetArgStr (Revision 2.0)
Description: gets the arguments for the current process
Library: dos.library
Offset: -$216
Syntax: ptr = GetArgStr( )
C: STRPTR GetArgStr(VOID)
ML: d0 = GetArgStr( )
Arguments: none
Result: ptr = pointer to arguments

GetAttr (Revision 2.0)
Description: gets the value of an object's attribute
Library: intuition.library
Offset: -$28E
Syntax: attr = GetAttr(attrID, object, storagePtr)
C: ULONG GetAttr(ULONG, APTR, ULONG *)
ML: d0 = GetAttr(d0, a0, a1)
Arguments: attrID = attribute tag ID
object = abstract pointer to the boopsi object
storagePtr = a buffer appropriate to receive the data
Result: attr = zero if unsuccessful

GetBitMapAttr (Revision 3.0)
Description: returns information about a bitmap
Library: graphics.library
Offset: -$3C0
Syntax: value = GetBitMapAttr(bitmap, attribute);
C: ULONG GetBitMapAttr(struct BitMap *, ULONG)
ML: d0 = GetBitMapAttr(a0, d1)
Arguments: bitMap = BitMap structure
attribute = information desired—BMA_HEIGHT ($00000000), BMA_WIDTH ($00000004), BMA_DEPTH ($00000008), BMA_FLAGS ($0000000C)
Result: value = bitmap height, width, depth, or flag info

GetBPen (Revision 3.0)
Description: returns a RastPort's APen value
Library: graphics.library
Offset: -$360
Syntax: pen = GetBPen (rastPort)
C: ULONG GetBPen(struct RastPort *)

ML: d0 = GetBPen (a0)
Arguments: rastPort = RastPort structure in question
Result: none

GetCatalogStr (Revision 3.0)
Description: retrieves a string from a message catalog
Library: locale.library
Offset: -$48
Syntax: string = GetCatalogStr(catalog, stringNum, defaultString)
C: STRPTR GetCatalogStr(struct Catalog *, LONG, STRPTR)
ML: d0 = GetCatalogStr(a0, d0, a1)
Arguments: catalog = message catalog returned by OpenCatalog( )
stringNum = number of the message to retrieve
defaultString = string to return in case of error
Result: string = the string message requested; read-only

GetCC
Description: returns the condition codes in a 68010-compatible way
Library: exec.library
Offset: -$210
Syntax: conditions = GetCC( )
C: UWORD GetCC(VOID)
ML: d0 = GetCC( )
Arguments: none
Result: conditions = the 680XX condition codes

GetColorMap
Description: allocates and initializes a ColorMap structure
Library: graphics.library
Offset: -$23A
Syntax: colorMap = GetColorMap(entries)
C: struct ColorMap *GetColorMap(ULONG)
ML: d0 = GetColorMap(d0)
Arguments: entries = number of color entries
Result: colorMap = pointer to allocated ColorMap; zero if unsuccessful

GetConsoleTask (Revision 2.0)
Description: returns the default console for the current process
Library: dos.library
Offset: -$1FE
Syntax: port = GetConsoleTask( )
C: struct MsgPort *GetConsoleTask(VOID)
ML: d0 = GetConsoleTask( )
Arguments: none
Result: port = pr_MsgPort of the console handler

GetCurrentBinding

Description: retrieves an expansion board's private configuration data
Library: expansion.library
Offset: -$8A
Syntax: actual = GetCurrentBinding(currentBinding, size)
C: ULONG GetCurrentBinding(struct CurrentBinding *, ULONG)
ML: d0 = GetCurrentBinding(a0, d0)
Arguments: currentBinding = a CurrentBinding structure to receive information
size = size of data to retrieve
Result: actual = the true size of the data available

GetCurrentDirName (Revision 2.0)

Description: returns the current directory name
Library: dos.library
Offset: -$234
Syntax: success = GetCurrentDirName(buf, len)
C: BOOL GetCurrentDirName(STRPTR, LONG)
ML: d0 = GetCurrentDirName(d1, d2)
Arguments: buf = buffer to hold extracted name
len = size of buffer
Result: success = zero if unsuccessful

GetDefaultPubScreen (Revision 2.0)

Description: gets the name of the default public screen
Library: intuition.library
Offset: -$246
Syntax: GetDefaultPubScreen(namebuff)
C: VOID GetDefaultPubScreen(UBYTE *)
ML: GetDefaultPubScreen(a0)
Arguments: namebuff = a 139-byte buffer to receive screen name
Result: none

GetDefDiskObject (Revision 2.0)

Description: reads a default workbench disk object from disk
Library: icon.library
Offset: -$78
Syntax: diskobj = GetDefDiskObject(defType)
C: struct DiskObject *GetDefDiskObject(LONG)
ML: d0 = GetDefDiskObject(d0)
Arguments: defType = default icon type—WBDISK ($00000001), WBDRAWER ($00000002), WBTOOL ($00000003), WBPROJECT ($00000004), WBGARBAGE ($00000005), WBKICK ($00000007)
Result: diskobj = the DiskObject structure found

GetDefPrefs

Description: gets a copy of the default Preferences structure
Library: intuition.library
Offset: -$7E
Syntax: Prefs = GetDefPrefs(prefBuffer, size)
C: struct Preferences *GetDefPrefs(struct Preferences *, WORD)
ML: d0 = GetDefPrefs(a0, d0)
Arguments: prefBuffer = buffer to receive data
size = size of buffer
Result: none

GetDeviceProc (Revision 2.0)

Description: finds a handler to send a message to
Library: dos.library
Offset: -$282
Syntax: devproc = GetDeviceProc(name, devproc)
C: struct DevProc *GetDeviceProc(STRPTR, struct DevProc *)
ML: d0 = GetDeviceProc(d1, d2)
Arguments: name = name of the object you wish to access
devproc = value returned by GetDeviceProc( ), or NULL
Result: devproc = a DevProc structure; zero if unsuccessful

GetDiskObject

Description: reads a Workbench disk object from disk
Library: icon.library
Offset: -$4E
Syntax: diskobj = GetDiskObject(name)
C: struct DiskObject *GetDiskObject(char *)
ML: d0 = GetDiskObject(a0)
Arguments: name = name of the object to load; 0 to allocate an empty object
Result: diskobj = the Workbench disk object requested

GetDiskObjectNew (Revision 2.0)

Description: reads a Workbench disk object from disk
Library: icon.library
Offset: -$84
Syntax: diskobj = GetDiskObjectNew(name)
C: struct DiskObject *GetDiskObjectNew(char *)
ML: d0 = GetDiskObjectNew(a0)
Arguments: name = name of the object to load; 0 to allocate an empty object GetDefDiskObject( ) is called if the object is not found
Result: diskobj = the Workbench disk object requested

GetDisplayInfoData (Revision 2.0)

Description: gets DisplayInfo record parameters
Library: graphics.library
Offset: -$2F4
Syntax: result = GetDisplayInfoData(handle, buf, size, tagID, [id])
C: ULONG GetDisplayInfoData(DisplayInfoHandle, UBYTE *, ULONG, ULONG, ULONG)
ML: d0 = GetDisplayInfoData(a0, a1, d0, d1, [d2])
Arguments: handle = DisplayInfo handle
buf = destination buffer
size = buffer size in bytes
tagID = data chunk type
id = displayinfo identifier; used only if handle argument is 0
Result: result = number of bytes transfered into destination buffer

GetDrMd (Revision 3.0)

Description: gets a RastPort's draw mode value
Library: graphics.library
Offset: -$366
Syntax: mode = GetDrMd(rastPort)
C: ULONG GetDrMd(struct RastPort *)
ML: d0 = GetDrMd(a0)
Arguments: rastPort = RastPort structure
Result: mode = draw mode

GetDTAttrsA (Revision 3.0)

Description: retrieves a DataType object's attributes
Library: datatypes.library
Offset: -$42
Syntax: retval = GetDTAttrsA(object, tags)
C: ULONG GetDTAttrsA(Object *, struct TagItem *)
ML: d0 = GetDTAttrsA(a0, a2)
Arguments: object = Object structure returned by NewDTObjectA( )
tags = tag list specifying which attributes to retrieve; data elements contain pointers to memory in which the retrieved data is to be stored
Result: retval = number of attributes retrieved

GetDTMethods (Revision 3.0)

Description: obtains list of methods supported by a DataType object
Library: datatypes.library
Offset: -$66
Syntax: methods = GetDTMethods(object)
C: ULONG GetDTMethods(Object *)
ML: d0 = GetDTMethods(a0)

Arguments: object = Object structure returned by NewDTObjectA( )
Result: methods = a zero-terminated array of ULONGs; becomes invalid when the object is disposed of

GetDTString (Revision 3.0)
Description: obtains access to a DataType string
Library: datatypes.library
Offset: -$8A
Syntax: str = GetDTString(id)
C: STRPTR GetDTString(ULONG id)
ML: d0 = GetDTString(d0)
Arguments: id = ID value of the string to access
Result: str = the string

GetDTTriggerMethods (Revision 3.0)
Description: obtains a list of trigger methods supported by a DataType object
Library: datatypes.library
Offset: -$6C
Syntax: methods = GetDTTriggerMethods(object)
C: struct DTMethods *GetDTTriggerMethods(Object *)
ML: d0 = GetDTTriggerMethods(a0)
Arguments: object = Object structure returned by NewDTObjectA( )
Result: methods = a zero-terminated array of ULONGs; becomes invalid when the object is disposed of

GetExtSpriteA (Revision 3.0)
Description: gets an extended sprite
Library: graphics.library
Offset: -$3A2
Syntax: spriteNum = GetExtSpriteA(sprite, tags)
C: spritenum = GetExtSprite(struct ExtSprite *, struct TagItem * tags)
ML: d0 = GetExtSpriteA(a2, a1)
Arguments: sprite = ExtSprite structure
tags = tag list of attributes—GSTAG_SPRITE_NUM ($82000020)
Result: spriteNum = sprite number; -1 if unsuccessful

GetFileSysTask (Revision 2.0)
Description: returns the default filesystem for the current process
Library: dos.library
Offset: -$20A
Syntax: port = GetFileSysTask( )
C: struct MsgPort *GetFileSysTask(VOID)
ML: d0 = GetFileSysTask( )
Arguments: none
Result: port = pr_MsgPort of the filesystem; zero if unsuccessful

GetGBuffers

Description: allocates all AnimOb buffers
Library: graphics.library
Offset: -$A8
Syntax: success = GetGBuffers(anOb, rastPort, db)
C: BOOL GetGBuffers(struct AnimOb *, struct RastPort *, BOOL)
ML: d0 = GetGBuffers(a0, a1, d0)
Arguments: anOb = AnimOb structure
rastPort = current RastPort
db = nonzero if you're double-buffering (Revision 3.0 only)
Result: success = zero if unsuccessful

GetLocaleStr (Revision 3.0)

Description: retrieves a standard string from a locale
Library: locale.library
Offset: -$4E
Syntax: string = GetLocaleStr(locale, stringNum)
C: STRPTR GetLocaleStr(struct Locale *, ULONG)
ML: d0 = GetLocaleStr(a0, d0)
Arguments: locale = Locale structure returned by OpenLocale( )
stringNum = the number of the string to retrieve—see <libraries locale.h> for description
Result: string = the string message requested; read-only, zero if unsuccessful

GetMsg

Description: gets the next message from a message port
Library: exec.library
Offset: -$174
Syntax: message = GetMsg(port)
C: struct Message *GetMsg(struct MsgPort *)
ML: d0 = GetMsg(a0)
Arguments: port = the message port which received a message
Result: message = the first message; zero if none are available—callers must be prepared for zero at any time

GetOPen (Revision 3.0)

Description: returns the OPen value for a RastPort
Library: graphics.library
Offset: -$36C
Syntax: pen = GetOPen(rastPort)
C: ULONG GetOPen(struct RastPort *)
ML: d0 = GetOPen(a0)
Arguments: rastPort = RastPort structure
Result: pen = OPen value

GetPrefs

Description: gets a copy of the current Preferences structure
Library: intuition.library
Offset: -$84
Syntax: Prefs = GetPrefs(prefBuffer, size)
C: struct Preferences *GetPrefs(struct Preferences *, WORD)
ML: d0 = GetPrefs(a0, d0)
Arguments: prefBuffer = buffer to receive data
size = size of buffer
Result: none

GetProgramDir

Description: returns a lock on the directory from which the current program was run
Library: dos.library
Offset: -$258
Syntax: lock = GetProgramDir( )
C: BPTR GetProgramDir(VOID)
ML: d0 = GetProgramDir( )
Arguments: none
Result: lock = program's directory; NULL if program was resident when run

GetProgramName (Revision 2.0)

Description: returns the current program name
Library: dos.library
Offset: -$240
Syntax: success = GetProgramName(buf, len)
C: BOOL GetProgramName(STRPTR, LONG)
ML: d0 = GetProgramName(d1, d2)
Arguments: buf = buffer to hold extracted name
len = size of buffer
Result: success = zero if unsuccessful

GetPrompt (Revision 2.0)

Description: returns the prompt for the current process
Library: dos.library
Offset: -$24C
Syntax: success = GetPrompt(buf, len)
C: BOOL GetPrompt(STRPTR, LONG)
ML: d0 = GetPrompt(d1, d2)
Arguments: buf = buffer to hold extracted prompt
len = size of buffer
Result: success = zero if unsuccessful

GetRGB32 (Revision 3.0)

Description: extracts 32-bit color values from a ColorMap structure
Library: graphics.library
Offset: -$384
Syntax: GetRGB32(colorMap, firstcolor, ncolors, table)
C: VOID GetRGB32(struct ColorMap *, ULONG, ULONG, ULONG *)
ML: GetRGB32(a0, d0, d1, a1)
Arguments: colorMap = colormap containing the desired colors
firstcolor = first color register to get
ncolors = number of color registers to read
table = buffer to receive extracted colors
Result: none

GetRGB4

Description: extracts a 12-bit color from a ColorMap structure
Library: graphics.library
Offset: -$246
Syntax: value = GetRGB4(colorMap, entry)
C: ULONG GetRGB4(struct ColorMap *, LONG)
ML: d0 = GetRGB32(a0, d0)
Arguments: colorMap = colormap containing the desired color
entry = index into colormap specifying desired entry
Result: value = 12-bit color value; -1 if no valid entry

GetRPAttrA (Revision 3.0)

Description: returns information about a RastPort
Library: graphics.library
Offset: -$414
Syntax: GetRPAttrA(rp, tags)
C: VOID GetRPAttrA(struct RastPort *, struct TagItem *)
ML: GetRPAttrA(a0, a1)
Arguments: rastPort = RastPort to examine
tags = tag list of attributes—RPTAG_Font ($80000000), RPTAG_APen ($80000002), RPTAG_BPen ($80000003), RPTAG_DrMd ($80000004), RPTAG_OutLinePen ($80000005), RPTAG_WriteMask ($80000006), RPTAG_MaxPen ($80000007)
Result: none

GetScreenData

Description: obtains all or part of a Screen structure, opening the screen if necessary
Library: intuition.library
Offset: -$1AA
Syntax: success = GetScreenData(buffer, size, type, screen)
C: BOOL GetScreenData(APTR, UWORD, UWORD, struct Screen *)
ML: d0 = GetScreenData(a0, d0, d1, a1)

Arguments: buffer =buffer to receive data
size = size of buffer
type = screen type, as specified in OpenWindow( ):
WBENCHSCREEN ($00000001), PUBLICSCREEN ($00000002),
CUSTOMSCREEN ($0000000F)
screen = custom Screen structure; ignored if type is not CUSTOMSCREEN
Result: success = zero if unsuccessful

GetScreenDrawInfo (Revision 2.0)

Description: obtains a screen's rendering information
Library: intuition.library
Offset: -$2BC
Syntax: drInfo = GetScreenDrawInfo(screen)
C: struct DrawInfo *GetScreenDrawInfo(struct Screen *)
ML: d0 = GetScreenDrawInfo(a0)
Arguments: screen = screen to examine
Result: drInfo = a system-allocated DrawInfo structure

GetSprite

Description: gets a simple sprite
Library: graphics.library
Offset: -$198
Syntax: spriteNum = GetSprite(sprite, pick)
C: WORD GetSprite(struct SimpleSprite *, WORD)
ML: d0 = GetSprite(a0, d0)
Arguments: sprite = sprite definition
pick = desired sprite number; 0-7, or -1 if you don't care
Result: spriteNum = sprite number of sprite you got; -1 if unsuccessful

GetTagData (Revision 2.0)

Description: obtains the data associated with a tag value
Library: utility.library
Offset: -$24
Syntax: value = GetTagData(tagValue, defaultVal, tags)
C: ULONG GetTagData(tagValue, ULONG, struct TagItem *)
ML: d0 = GetTagData(d0, d1, a0)
Arguments: tagValue = tag value to search for
defaultVal = value to be returned if tagValue is not found
tags = the tag list to search
Result: value = data in TagItem matching tagValue if found, defaultVal if
not found

GetUniqueID (Revision 3.0)

Description: returns a relatively unique number
Library: utility.library
Offset: -$10E
Syntax: ID = GetUniqueID( )
C: ULONG GetUniqueID(VOID)
ML: d0 = GetUniqueID( )
Arguments: none
Result: ID = a unique 32-bit value

GetVar (Revision 2.0)

Description: returns the value of an environment variable
Library: dos.library
Offset: -$38A
Syntax: len = GetVar(name, buffer, size, flags)
C: LONG GetVar(STRPTR, STRPTR, LONG, ULONG)
ML: d0 = GetVar(d1, d2, d3, d4)
Arguments: name = variable name
buffer = buffer to hold value
size = size of buffer
flags = type of variable to get:
GVF_GLOBAL_ONLY ($00000100), GVF_LOCAL_ONLY ($00000200),
GVF_BINARY_VAR ($00000400)
Result: len = size of environment variable; -1 if unsuccessful

GetVisualInfoA (Revision 2.0)

Description: gets information for GadTools display
Library: gadtools.library
Offset: -$7E
Syntax: vi = GetVisualInfoA(screen, tags)
C: APTR vi = GetVisualInfo(struct Screen *, struct TagItem *)
ML: d0 = GetVisualInfoA(a0, a1)
Arguments: screen = the screen on which GadTools will operate
tags = tag list of attributes—none are currently defined
Result: vi = pointer to private data; zero if unsuccessfull

GetVPModeID (Revision 2.0)

Description: gets a ViewPort's 32-bit DisplayID
Library: graphics.library
Offset: -$318
Syntax: modeID = GetVPModeID(viewPort)
C: ULONG GetVPModeID(struct ViewPort *)
ML: d0 = GetVPModeID(a0)

Arguments: viewPort = ViewPort structure
Result: modeID = ViewPort's 32-bit DisplayInfoRecord identifier; -1 if unsuccessful

GfxAssociate (Revision 2.0)
Description: associates a graphics extended node with a given pointer
Library: graphics.library
Offset: -$2A0
Syntax: GfxAssociate(pointer, node)
C: VOID GfxAssociate(VOID *, struct ExtendedNode *)
ML: GfxAssociate(a0, a1)
Arguments: pointer = data structure
node = ExtendedNode structure to associate with the pointer
Result: none

GfxFree (Revision 2.0)
Description: frees a graphics extended data structure
Library: graphics.library
Offset: -$29A
Syntax: GfxFree(node)
C: VOID GfxFree(struct ExtendedNode *)
ML: GfxFree(a0)
Arguments: node = graphics extended data structure returned by GfxNew( ) function
Result: none

GfxLookUP (Revision 2.0)
Description: finds a graphics extended node associated with a pointer
Library: graphics.library
Offset: -$2BE
Syntax: result = GfxLookUp(pointer)
C: struct ExtendedNode *GfxLookUp(VOID *)
ML: d0 = GfxLookUp(a0)
Arguments: pointer = pointer in question
Result: result = associated ExtendedNode structure; zero if unsuccessful

GfxNew (Revision 2.0)
Description: allocates a graphics extended data structure
Library: graphics.library
Offset: -$294
Syntax: node = GfxNew(nodeType)
C: struct ExtendedNode *GfxNew(ULONG)
ML: d0 = GfxNew(d0)

Arguments: nodeType = type of graphics extended data structure to allocate:
VIEW_EXTRA_TYPE ($00000001), VIEWPORT_EXTRA_TYPE ($00000002),
SPECIAL_MONITOR_TYPE ($00000003), MONITOR_SPEC_TYPE ($00000004)
Result: node = extended data structure; zero if unsuccessful

GoodID (Revision 2.0)

Description: tests an identifier for adherance to the IFF 85 specification
Library: iffparse.library
Offset: -$102
Syntax: isok = GoodID(id)
C: LONG GoodID(LONG)
ML: d0 = GoodID(d0)
Arguments: id = 32-bit identifier
Result: isok = zero if the id is invalid

GoodType (Revision 2.0)

Description: tests a type for adherance to the IFF 85 specification
Library: iffparse.library
Offset: -$108
Syntax: isok = GoodType(type)
C: LONG GoodType(LONG)
ML: d0 = GoodType(d0)
Arguments: type = 32-bit format type identifier
Result: isok = zero if the type is invalid

GT_BeginRefresh (Revision 2.0)

Description: begins a GadTools-friendly refresh
Library: gadtools.library
Offset: -$5A
Syntax: GT_BeginRefresh(win)
C: VOID GT_BeginRefresh(struct Window *)
ML: GT_BeginRefresh(a0)
Arguments: win = a Window structure to which GadTools gadgets are attached
Result: none

GT_EndRefresh (Revision 2.0)

Description: ends a GadTools-friendly refresh
Library: gadtools.library
Offset: -$60
Syntax: GT_EndRefresh(win, complete)
C: VOID GT_EndRefresh(struct Window *, BOOL)
ML: GT_EndRefresh(a0, d0)
Arguments: win = a Window structure to which GadTools gadgets are attached
complete = nonzero when done with refresh
Result: none

GT_FilterIMsg (Revision 2.0)

Description: filters an IntuiMessage through GadTools
Library: gadtools.library
Offset: -$66
Syntax: modimsg = GT_FilterIMsg(msg)
C: struct IntuiMessage *GT_FilterIMsg(struct IntuiMessage *)
ML: d0 = GT_FilterIMsg(a1)
Arguments: imsg = IntuiMessage structure to pass to GadTools
Result: modimsg = possibly-modified IntuiMessage structure;
zero if not a GadTools message and should be returned with ReplyMsg( )

GT_GetGadgetAttrsA (Revision 3.0)

Description: obtains the attributes of a GadTools gadget
Library: gadtools.library
Offset: -$AE
Syntax: numProcessed = GT_GetGadgetAttrsA(gad, win, req, tags)
C: LONG GT_GetGadgetAttrsA(struct Gadget *, struct Window *, struct Requester *,
struct TagItem *)
ML: d0 = GT_GetGadgetAttrsA(a0, a1, a2, a3)
Arguments: gad = the gadget to obtain information about
win = the window containing the gadget.
req = reserved for future use; should be zero
tags = tag list to receive data:
see <libraries/gadtools.h> for description
Result: numProcessed = the number of data actually retrieved

GT_GetIMsg (Revision 2.0)

Description: retrieves an IntuiMessage with Gadtools processing
Library: gadtools.library
Offset: -$48
Syntax: imsg = GT_GetIMsg(msgport)
C: struct IntuiMessage *GT_GetIMsg(struct MsgPort *)
ML: d0 = GT_GetIMsg(a0)
Arguments: msgport = MsgPort of a Window that has GadTools gadgets attached
Result: imsg = pointer to Intuimessage structure;
zero if no messages are available or if all available messages relate
only to GadTools

GT_PostFilterIMsg (Revision 2.0)

Description: returns an unfiltered message which has been passed to GT_FilterIMsg( )
Library: gadtools.library
Offset: -$6C
Syntax: imsg = GT_PostFilterIMsg(modimsg)
C: struct IntuiMessage *GT_PostFilterIMsg(struct IntuiMessage *)

ML: d0 = GT_PostFilterIMsg(a1)
Arguments: modimsg = IntuiMessage structure returned by GT_FilterIMsg( )
Result: imsg = original IntuiMessage structure passed to GT_FilterIMsg( )

GT_RefreshWindow (Revision 2.0)

Description: performs initial refresh of GadTools gadgets in a window
Library: gadtools.library
Offset: -$54
Syntax: GT_RefreshWindow(win, req)
C: VOID GT_RefreshWindow(struct Window *, struct Requester *)
ML: GT_RefreshWindow(a0, a1)
Arguments: win = the Window containing the GadTools gadgets
req = reserved for future use; should be zero
Result: none

GT_ReplyIMsg (Revision 2.0)

Description: replies to a message obtained with GT_GetIMsg( )
Library: gadtools.library
Offset: -$4E
Syntax: GT_ReplyIMsg(imsg)
C: VOID GT_ReplyIMsg(struct IntuiMessage *)
ML: GT_ReplyIMsg(a1)
Arguments: imsg = IntuiMessage structure returned by GT_GetIMsg( )
Result: none

GT_SetGadgetAttrsA (Revision 2.0)

Description: sets attributes of a GadTools gadget
Library: gadtools.library
Offset: -$2A
Syntax: GT_SetGadgetAttrsA(gad, win, req, tags)
C: VOID GT_SetGadgetAttrsA(struct Gadget *, struct Window *, struct Requester *, struct TagItem *)
ML: GT_SetGadgetAttrsA(a0, a1, a2, a3)
Arguments: gad = the gadget to modify
win = the window containing the gadget
req = reserved for future use; should be zero
tags = tag list of parameters to set—see <libraries/gadtools.h> for description
Result: none

HelpControl (Revision 3.0)

Description: enables or disables Gadget-Help feature
Library: intuition.library
Offset: -$33C
Syntax: HelpControl(window, flags)
C: VOID HelpControl(struct Window *, ULONG)
ML: HelpControl(a0, d0)
Arguments: window = window to affect
flags = HC_GADGETHELP ($00000001) to enable Gadget-Help, 0 to disable Gadget-Help
Result: none

IDtoStr (Revision 2.0)

Description: converts a longword identifier to a zero-terminated string
Library: iffparse.library
Offset: -$10E
Syntax: str = IDtoStr(id, buf)
C: STRPTR IDtoStr(LONG, STRPTR)
ML: d0 = IDtoStr(d0, a0)
Arguments: id = longword id
buf = character buffer to receive string; must be at least five characters long
Result: str = buf filled with zero-terminated representation of id

IEEEDPAbs

Description: computes the absolute value of an IEEE double precision number
Library: mathffp.library
Offset: -$36
Syntax: fnum2 = IEEEDPAbs(fnum1)
C: double IEEEDPAbs(double)
ML: d0/d1 = IEEEDPAbs(d0/d1)
Arguments: fnum1 = number for which to find absolute value
Result: fnum2 = absolute value of fnum1

IEEEDPAcos

Description: computes the arccosine of an IEEE double precision number
Library: mathtrans.library
Offset: -$78
Syntax: fnum2 = IEEEDPAcos(fnum1)
C: double IEEEDPAcos(double)
ML: d0/d1 = IEEEDPAcos(d0/d1)
Arguments: fnum1 = number for which to compute arccosine
Result: fnum2 = arccosine of fnum1

IEEEDPAdd

Description: computes the sum of two IEEE double precision numbers
Library: mathffp.library
Offset: -$42
Syntax: fnum3 = IEEEDPAdd(fnum1, fnum2)
C: double IEEEDPAdd(double, double)
ML: d0/d1 = IEEEDPAdd(d0/d1, d2/d3)
Arguments: fnum1, fnum2 = summands
Result: fnum3 = value of fnum1 + fnum2.

IEEEDPAsin

Description: computes the arcsine of an IEEE double precision number
Library: mathtrans.library
Offset: -$72
Syntax: fnum2 = IEEEDPAsin(fnum1)
C: double IEEEDPAsin(double)
ML: d0/d1 = IEEEDPAsin(d0/d1)
Arguments: fnum1 = number for which to compute arcsine
Result: fnum2 = arcsine of fnum1

IEEEDPAtan

Description: computes the arctangent of an IEEE double precision number
Library: mathtrans.library
Offset: -$1E
Syntax: fnum2 = IEEEDPAtan(fnum1)
C: double IEEEDPAtan(double)
ML: d0/d1 = IEEEDPAtan(d0/d1)
Arguments: fnum1 = number for which to compute arctangent
Result: fnum2 = arctangent of fnum1

IEEEDPCeil

Description: computes the ceiling of an IEEE double precision number
Library: mathffp.library
Offset: -$60
Syntax: fnum2 = IEEEDPCeil(fnum1)
C: double IEEEDPCeil(double)
ML: d0/d1 = IEEEDPCeil(d0/d1)
Arguments: fnum1 = number for which to compute ceiling
Result: fnum2 = smallest whole number greater than or equal to fnum1

IEEEDPCmp

Description: compares two IEEE double precision numbers and sets condition codes
Library: mathffp.library
Offset: -$2A
Syntax: result = IEEEDPCmp(fnum1, fnum2)

C: int IEEEDPCmp(double, double)
ML: d0 = IEEEDPCmp(d0/d1, d2/d3)
Arguments: fnum1, fnum2 = numbers to compare
Result: result = 1 if fnum1 > fnum2, zero if fnum1 equals fnum2, -1 if fnum1 < fnum2

IEEEDPCos

Description: computes the cosine of an IEEE double precision number
Library: mathtrans.library
Offset: -$2A
Syntax: fnum2 = IEEEDPCos(fnum1)
C: double IEEEDPCos(double)
ML: d0/d1 = IEEEDPCos(d0/d1)
Arguments: fnum1 = number for which to compute cosine
Result: fnum2 = cosine of fnum1

IEEEDPCosh

Description: computes the hyperbolic cosine of an IEEE double precision number
Library: mathtrans.library
Offset: -$42
Syntax: fnum2 = IEEEDPCosh(fnum1)
C: double IEEEDPCosh(double)
ML: d0/d1 = IEEEDPCosh(d0/d1)
Arguments: fnum1 = number for which to compute hyperbolic cosine
Result: fnum2 = hyperbolic cosine of fnum1

IEEEDPDiv

Description: computes the quotient of two IEEE double precision numbers
Library: mathffp.library
Offset: -$54
Syntax: fnum3 = IEEEDPDiv(fnum1, fnum2)
C: double IEEEDPDiv(double, double)
ML: d0/d1 = IEEEDPDiv(d0/d1, d2/d3)
Arguments: fnum1 = dividend
fnum2 = divisor
Result: fnum3 = value of fnum1 / fnum2

IEEEDPExp

Description: computes the exponential of an IEEE double precision number
Library: mathtrans.library
Offset: -$4E
Syntax: fnum2 = IEEEDPExp(fnum1)
C: double IEEEDPExp(double)
ML: d0/d1 = IEEEDPExp(d0/d1)

Arguments: fnum1 = number for which to compute the exponential
Result: fnum2 = exponential of fnum1

IEEEDPFieee

Description: converts an IEEE single precision number to an IEEE double precision number
Library: mathtrans.library
Offset: -$6C
Syntax: fnum = IEEEDPFieee(ieeenum)
C: double IEEEDPFieee(float)
ML: d0/d1 = IEEEDPFieee(d0)
Arguments: ieeenum = IEEE single precision number to convert
Result: fnum = IEEE double precision representation of ieeenum

IEEEDPFix

Description: converts an IEEE double precision number to an integer
Library: mathffp.library
Offset: -$1E
Syntax: inum = IEEEDPFix(fnum)
C: int IEEEDPFix(double)
ML: d0 = IEEEDPFix(d0/d1)
Arguments: fnum = number to convert
Result: inum = truncated integer value of fnum

IEEEDPFloor

Description: computes the floor of a IEEE double precision number
Library: mathffp.library
Offset: -$5A
Syntax: fnum2 = IEEEDPFloor(fnum1)
C: double IEEEDPFloor(double)
ML: d0/d1 = IEEEDPFloor(d0/d1)
Arguments: fnum1 = number for which to compute floor
Result: fnum2 = largest whole number less than or equal to fnum1

IEEEDPFlt

Description: converts an integer to an IEEE double precision number
Library: mathffp.library
Offset: -$24
Syntax: fnum = IEEEDPFlt(inum)
C: double IEEEDPFlt(int)
ML: d0/d1 = IEEEDPFlt(d0)
Arguments: inum = integer to convert
Result: fnum = floating point representation of inum

IEEEDPLog

Description: computes the natural logarithm of an IEEE double precision number
Library: mathtrans.library
Offset: -$54
Syntax: fnum2 = IEEEDPLog(fnum1)
C: double IEEEDPLog(double)
ML: d0/d1 = IEEEDPLog(d0/d1)
Arguments: fnum1 = number for which to compute natural logarithm
Result: fnum2 = natural logarithm of fnum1

IEEEDPLog10

Description: computes the base 10 logarithm of an IEEE double precision number
Library: mathtrans.library
Offset: -$7E
Syntax: fnum2 = IEEEDPLog10(fnum1)
C: double IEEEDPLog10(double)
ML: d0/d1 = IEEEDPLog10(d0/d1)
Arguments: fnum1 = number for which to compute logarithm
Result: fnum2 = base 10 logarithm of fnum1

IEEEDPMul

Description: computes the product of two IEEE double precision numbers
Library: mathffp.library
Offset: -$4E
Syntax: fnum3 = IEEEDPMul(fnum1, fnum2)
C: double IEEEDPMul(double, double)
ML: d0/d1 = IEEEDPMul(d0/d1, d2/d3)
Arguments: fnum1 = multiplicand
fnum2 = multiplier
Result: fnum3 = value of fnum1 * fnum2

IEEEDPNeg

Description: negates an IEEE double precision number
Library: mathffp.library
Offset: -$3C
Syntax: fnum2 = IEEEDPNeg(fnum1)
C: double IEEEDPNeg(double)
ML: d0/d1 = IEEEDPNeg(d0/d1)
Arguments: fnum1 = number to negate
Result: fnum2 = -fnum1

IEEEDPPow

Description: raises a number to a power
Library: mathtrans.library
Offset: -$5A

Syntax: result = IEEEDPPow(fnum1, fnum2)
C: double = IEEEDPPow(double, double)
ML: d0/d1 = IEEEDPPow(d2/d3, d0/d1)
Arguments: fnum1 = exponent to fnum2
fnum2 = value to raise to the power fnum1
Result: result = value of fnum2 raised to the fnum1 power

IEEEDPSin

Description: computes the sine of an IEEE double precision number
Library: mathtrans.library
Offset: -$24
Syntax: fnum2 = IEEEDPSin(fnum1)
C: double IEEEDPSin(double)
ML: d0/d1 = IEEEDPSin(d0/d1)
Arguments: fnum1 = number for which to compute sine
Result: fnum2 = sine of fnum1

IEEEDPSincos

Description: computes sine and cosine of an IEEE double precision number
Library: mathtrans.library
Offset: -$36
Syntax: fnum3 = IEEEDPSincos(pfnum2, fnum1)
C: double = IEEEDPSincos(double *, double)
ML: d0/d1 = IEEEDPSincos(a0, d0/d1)
Arguments: fnum1 = number for which to compute sine and cosine
pfnum2 = buffer to receive the cosine of fnum1
Result: fnum3 = sine of fnum1

IEEEDPSinh

Description: computes the hyperbolic sine of an IEEE double precision number
Library: mathtrans.library
Offset: -$3C
Syntax: fnum2 = IEEEDPSinh(fnum1)
C: double IEEEDPSinh(double)
ML: d0/d1 = IEEEDPSinh(d0/d1)
Arguments: fnum1 = number for which to compute hyperbolic sine
Result: fnum2 = hyperbolic sine of fnum1

IEEEDPSqrt

Description: computes the square root of an IEEE double precision number
Library: mathtrans.library
Offset: -$60
Syntax: fnum2 = IEEEDPSqrt(fnum1)
C: double IEEEDPSqrt(double)
ML: d0/d1 = IEEEDPSqrt(d0/d1)

Arguments: fnum1 = number for which to compute square root
Result: fnum2 = square root of fnum1

IEEEDPSub
Description: computes the difference between two IEEE double precision numbers
Library: mathffp.library
Offset: -$48
Syntax: fnum3 = IEEEDPSub(fnum1, fnum2)
C: double IEEEDPSub(double, double)
ML: d0/d1 = IEEEDPSub(d0/d1, d2/d3)
Arguments: fnum1 = minuend
fnum2 = subtrahend
Result: fnum3 = value of fnum1 - fnum2

IEEEDPTan
Description: computes the tangent of an IEEE double precision number
Library: mathtrans.library
Offset: -$30
Syntax: fnum2 = IEEEDPTan(fnum1)
C: double IEEEDPTan(double)
ML: d0/d1 = IEEEDPTan(d0/d1)
Arguments: fnum1 = number for which to compute tangent
Result: fnum2 = tangent of fnum1

IEEEDPTanh
Description: computes the hyperbolic tangent of an IEEE double precision number
Library: mathtrans.library
Offset: -$48
Syntax: fnum2 = IEEEDPTanh(fnum1)
C: double IEEEDPTanh(double)
ML: d0/d1 = IEEEDPTanh(d0/d1)
Arguments: fnum1 = number for which to compute hyperbolic tangent
Result: fnum2 = hyperbolic tangent of fnum1

IEEEDPTieee
Description: converts an IEEE double precision number to an IEEE single precision number
Library: mathtrans.library
Offset: -$66
Syntax: ieeenum = IEEEDPTieee(fnum)
C: float IEEEDPTieee(double)
ML: d0 = IEEEDPTieee(d0/d1)
Arguments: fnum = IEEE double precision number to convert
Result: ieeenum = IEEE single precision representation of fnum

IEEEDPTst

Description: compares an IEEE double precision number to zero (0.0)
Library: mathffp.library
Offset: -$30
Syntax: result = IEEEDPTst(fnum)
C: int IEEEDPTst(double)
ML: d0 = IEEEDPTst(d0/d1)
Arguments: fnum = number to compare to zero
Result: result = 1 if fnum1 > 0.0, zero if fnum1 = 0.0, -1 if fnum1 < 0.0

IEEESPAbs

Description: computes the absolute value of an IEEE single precision number
Library: mathffp.library
Offset: -$36
Syntax: fnum2 = IEEESPAbs(fnum1)
C: float IEEESPAbs(float)
ML: d0 = IEEESPAbs(d0)
Arguments: fnum1 = number for which to compute absolute value
Result: fnum2 = absolute value of fnum1

IEEESPAcos

Description: computes the arccosine of an IEEE single precision number
Library: mathtrans.library
Offset: -$78
Syntax: fnum2 = IEEESPAcos(fnum1)
C: float IEEESPAcos(float)
ML: d0 = IEEESPAcos(d0)
Arguments: fnum1 = number for which to compute arccosine
Result: fnum2 = arccosine of fnum1

IEEESPAdd

Description: computes the sum of two IEEE single precision numbers
Library: mathffp.library
Offset: -$42
Syntax: fnum3 = IEEESPAdd(fnum1, fnum2)
C: float IEEESPAdd(float, float)
ML: d0 = IEEESPAdd(d0, d1)
Arguments: fnum1, fnum2 = summands
Result: fnum3 = value of fnum1 + fnum2.

IEEESPAsin

Description: computes the arcsine of an IEEE single precision number
Library: mathtrans.library
Offset: -$72
Syntax: fnum2 = IEEESPAsin(fnum1)

C: float IEEESPAsin(float)
ML: d0 = IEEESPAsin(d0)
Arguments: fnum1 = number for which to compute arcsine
Result: fnum2 = arcsine of fnum1

IEEESPAtan
Description: computes the arctangent of an IEEE single precision number
Library: mathtrans.library
Offset: -$1E
Syntax: fnum2 = IEEESPAtan(fnum1)
C: float IEEESPAtan(float)
ML: d0 = IEEESPAtan(d0)
Arguments: fnum1 = number for which to compute arctangent
Result: fnum2 = arctangent of fnum1

IEEESPCeil
Description: computes the ceiling of an IEEE single precision number
Library: mathffp.library
Offset: -$60
Syntax: fnum2 = IEEESPCeil(fnum1)
C: float IEEESPCeil(float)
ML: d0 = IEEESPCeil(d0)
Arguments: fnum1 = number for which to compute ceiling
Result: fnum2 = smallest whole number greater than or equal to fnum1

IEEESPCmp
Description: compares two IEEE single precision numbers and sets condition codes
Library: mathffp.library
Offset: -$2A
Syntax: result = IEEESPCmp(fnum1, fnum2)
C: int IEEESPCmp(float, float)
ML: d0 = IEEESPCmp(d0, d1)
Arguments: fnum1, fnum2 = numbers to compare
Result: result = 1 if fnum1 > fnum2, zero if fnum1 = fnum2, -1 if fnum1 < fnum2

IEEESPCos
Description: computes the cosine of an IEEE single precision number
Library: mathtrans.library
Offset: -$2A
Syntax: fnum2 = IEEESPCos(fnum1)
C: float IEEESPCos(float)
ML: d0 = IEEESPCos(d0)
Arguments: fnum1 = number for which to compute cosine
Result: fnum2 = cosine of fnum1

IEEESPCosh

Description: computes the hyperbolic cosine of an IEEE single precision number
Library: mathtrans.library
Offset: -$42
Syntax: fnum2 = IEEESPCosh(fnum1)
C: float IEEESPCosh(float)
ML: d0 = IEEESPCosh(d0)
Arguments: fnum1 = number for which to compute hyperbolic cosine
Result: fnum2 = hyperbolic cosine of fnum1

IEEESPDiv

Description: computes the quotient of two IEEE single precision numbers
Library: mathffp.library
Offset: -$54
Syntax: fnum3 = IEEESPDiv(fnum1, fnum2)
C: float IEEESPDiv(float, float)
ML: d0 = IEEESPDiv(d0, d1)
Arguments: fnum1 = dividend
fnum2 = divisor
Result: fnum3 = value of fnum1 / fnum2

IEEESPExp

Description: computes the exponential of an IEEE single precision number
Library: mathtrans.library
Offset: -$4E
Syntax: fnum2 = IEEESPExp(fnum1)
C: float IEEESPExp(float)
ML: d0 = IEEESPExp(d0)
Arguments: fnum1 = number for which to compute the exponential
Result: fnum2 = exponential of fnum1

IEEESPFieee

Description: converts an IEEE single precision number to an IEEE single precision number
Library: mathtrans.library
Offset: -$6C
Syntax: fnum = IEEESPFieee(ieeenum)
C: float IEEESPFieee(float)
ML: d0 = IEEESPFieee(d0)
Arguments: ieeenum = IEEE single precision number to convert
Result: fnum = IEEE single precision representation of ieeenum

IEEESPFix

Description: converts an IEEE single precision number to an integer
Library: mathffp.library

Offset: -$1E
Syntax: inum = IEEESPFix(fnum)
C: int IEEESPFix(float)
ML: d0 = IEEESPFix(d0)
Arguments: fnum = number to convert
Result: inum = truncated integer value of fnum

IEEESPFloor
Description: computes the floor of a IEEE single precision number
Library: mathffp.library
Offset: -$5A
Syntax: fnum2 = IEEESPFloor(fnum1)
C: float IEEESPFloor(float)
ML: d0 = IEEESPFloor(d0)
Arguments: fnum1 = number for which to compute floor
Result: fnum2 = largest whole number less than or equal to fnum1

IEEESPFlt
Description: converts an integer to an IEEE single precision number
Library: mathffp.library
Offset: -$24
Syntax: fnum = IEEESPFlt(inum)
C: float IEEESPFlt(int)
ML: d0 = IEEESPFlt(d0)
Arguments: inum = integer to convert
Result: fnum = floating point representation of inum

IEEESPLog
Description: computes the natural logarithm of an IEEE single precision number
Library: mathtrans.library
Offset: -$54
Syntax: fnum2 = IEEESPLog(fnum1)
C: float IEEESPLog(float)
ML: d0 = IEEESPLog(d0)
Arguments: fnum1 = number for which to compute natural logarithm
Result: fnum2 = natural logarithm of fnum1

IEEESPLog10
Description: computes the base 10 logarithm of an IEEE single precision number
Library: mathtrans.library
Offset: -$7E
Syntax: fnum2 = IEEESPLog10(fnum1)
C: float IEEESPLog10(float)
ML: d0 = IEEESPLog10(d0)

Arguments: fnum1 = number for which to compute logarithm
Result: fnum2 = base 10 logarithm of fnum1

IEEESPMul

Description: computes the product of two IEEE single precision numbers
Library: mathffp.library
Offset: -$4E
Syntax: fnum3 = IEEESPMul(fnum1, fnum2)
C: float IEEESPMul(float, float)
ML: d0 = IEEESPMul(d0, d1)
Arguments: fnum1 = multiplicand
fnum2 = multiplier
Result: fnum3 = value of fnum1 * fnum2

IEEESPNeg

Description: negates an IEEE single precision number
Library: mathffp.library
Offset: -$3C
Syntax: fnum2 = IEEESPNeg(fnum1)
C: float IEEESPNeg(float)
ML: d0 = IEEESPNeg(d0)
Arguments: fnum1 = number to negate
Result: fnum2 = -fnum1

IEEESPPow

Description: raises a number to a power
Library: mathtrans.library
Offset: -$5A
Syntax: result = IEEESPPow(fnum1, fnum2)
C: float = IEEESPPow(float, float)
ML: d0 = IEEESPPow(d1, d0)
Arguments: fnum1 = exponent to fnum2
fnum2 = value to raise to the power fnum1
Result: result = value of fnum2 raised to the fnum1 power

IEEESPSin

Description: computes the sine of an IEEE single precision number
Library: mathtrans.library
Offset: -$24
Syntax: fnum2 = IEEESPSin(fnum1)
C: float IEEESPSin(float)
ML: d0 = IEEESPSin(d0)
Arguments: fnum1 = number for which to compute sine
Result: fnum2 = sine of fnum1

IEEESPSincos

Description: computes sine and cosine of an IEEE single precision number
Library: mathtrans.library
Offset: -$36
Syntax: fnum3 = IEEESPSincos(pfnum2, fnum1)
C: float = IEEESPSincos(float *, float)
ML: d0 = IEEESPSincos(a0, d0)
Arguments: fnum1 = number for which to compute sine and cosine
pfnum2 = buffer to receive the cosine of fnum1
Result: fnum3 = sine of fnum1

IEEESPSinh

Description: computes the hyperbolic sine of an IEEE single precision number
Library: mathtrans.library
Offset: -$3C
Syntax: fnum2 = IEEESPSinh(fnum1)
C: float IEEESPSinh(float)
ML: d0 = IEEESPSinh(d0)
Arguments: fnum1 = number for which to compute hyperbolic sine
Result: fnum2 = hyperbolic sine of fnum1

IEEESPSqrt

Description: computes the square root of an IEEE single precision number
Library: mathtrans.library
Offset: -$60
Syntax: fnum2 = IEEESPSqrt(fnum1)
C: float IEEESPSqrt(float)
ML: d0 = IEEESPSqrt(d0)
Arguments: fnum1 = number for which to compute square root
Result: fnum2 = square root of fnum1

IEEESPSub

Description: computes the difference between two IEEE single precision numbers
Library: mathffp.library
Offset: -$48
Syntax: fnum3 = IEEESPSub(fnum1, fnum2)
C: float IEEESPSub(float, float)
ML: d0 = IEEESPSub(d0, d1)
Arguments: fnum1 = minuend
fnum2 = subtrahend
Result: fnum3 = value of fnum1 - fnum2

IEEESPTan

Description: computes the tangent of an IEEE single precision number
Library: mathtrans.library

Offset: -$30
Syntax: fnum2 = IEEESPTan(fnum1)
C: float IEEESPTan(float)
ML: d0 = IEEESPTan(d0)
Arguments: fnum1 = number for which to compute tangent
Result: fnum2 = tangent of fnum1

IEEESPTanh

Description: computes the hyperbolic tangent of an IEEE single precision number
Library: mathtrans.library
Offset: -$48
Syntax: fnum2 = IEEESPTanh(fnum1)
C: float IEEESPTanh(float)
ML: d0 = IEEESPTanh(d0)
Arguments: fnum1 = number for which to compute hyperbolic tangent
Result: fnum2 = hyperbolic tangent of fnum1

IEEESPTieee

Description: converts an IEEE single precision number to an IEEE single precision number
Library: mathtrans.library
Offset: -$66
Syntax: ieeenum = IEEESPTieee(fnum)
C: float IEEESPTieee(float)
ML: d0 = IEEESPTieee(d0)
Arguments: fnum = IEEE single precision number to convert
Result: ieeenum = IEEE single precision representation of fnum

IEEESPTst

Description: compares an IEEE single precision number to zero (0.0)
Library: mathffp.library
Offset: -$30
Syntax: result = IEEESPTst(fnum)
C: int IEEESPTst(float)
ML: d0 = IEEESPTst(d0)
Arguments: fnum = number to compare to zero
Result: result = 1 if fnum1 > 0.0, zero if fnum1 = 0.0, -1 if fnum1 < 0.0

Info

Description: returns information about a disk
Library: dos.library
Offset: -$72
Syntax: success = Info(lock, parameterBlock)
C: BOOL Info(BPTR, struct InfoData *)
ML: d0 = Info(d1, d2)

Arguments: lock = lock on disk in question
parameterBlock = InfoData structure to receive information
Result: success = zero if unsuccessful

Inhibit (Revision 2.0)
Description: determines the accessibility of a filesystem
Library: dos.library
Offset: -$2D6
Syntax: success = Inhibit(filesystem, flag)
C: BOOL Inhibit(STRPTR, LONG)
ML: d0 = Inhibit(d1, d2)
Arguments: filesystem = name of device to inhibit (with :)
flag = desired status; -1 for inhibited, 0 for uninhibited
Result: success = zero if unsuccessful

InitArea
Description: initializes a vector collection matrix
Library: graphics.library
Offset: -$11A
Syntax: InitArea(areainfo, buffer, maxvectors)
C: VOID InitArea(struct AreaInfo *, VOID *, WORD)
ML: InitArea(a0, a1, d0)
Arguments: areainfo = AreaInfo structure
buffer = memory buffer to receive vertices; should be at least 5*maxvectors bytes in size
maxvectors = maximum number of vectors this buffer can hold
Result: none

InitBitMap
Description: initializes bitmap structure
Library: graphics.library
Offset: -$186
Syntax: InitBitMap(bitMap, depth, width, height)
C: VOID InitBitMap(struct BitMap *, BYTE, UWORD, UWORD)
ML: InitBitMap(a0, d0, d1, d2)
Arguments: bitMap = BitMap structure
depth = number of bitplanes
width, height = dimensions of the bitmap
Result: none

InitCode
Description: initializes resident code modules (internal function)
Library: exec.library
Offset: -$48
Syntax: InitCode(startClass, version)

C: VOID InitCode(ULONG, ULONG)
ML: InitCode(d0, d1)
Arguments: startClass = the class of code to be initialized
see <exec/resident.h> for description
version = a major version number
Result: none

InitGels

Description: initializes a gel list
Library: graphics.library
Offset: -$78
Syntax: InitGels(head, tail, gInfo)
C: VOID InitGels(struct VSprite *, struct VSprite *, struct GelsInfo *)
ML: InitGels(a0, a1, a2)
Arguments: head = VSprite structure to use as the gel list head
tail = VSprite structure to use as the gel list tail
gInfo = GelsInfo structure to be initialized
Result: none

InitGMasks

Description: initializes an AnimObs masks
Library: graphics.library
Offset: -$AE
Syntax: InitGMasks(anOb)
C: VOID InitGMasks(struct AnimOb *)
ML: InitGMasks(a0)
Arguments: anOb = AnimOb structure
Result: none

InitIFF (Revision 2.0)

Description: initializes an IFFHandle structure as a user stream
Library: iffparse.library
Offset: -$E4
Syntax: InitIFF(iff, flags, streamHook)
C: VOID InitIFF(struct IFFHandle *, LONG, struct Hook *)
ML: InitIFF(a0, d0, a1)
Arguments: iff = the IFFHandle structure to initialize
flags = stream I/O flags for the IFFHandle:
IFFF_READ ($00000000), IFFF_WRITE ($00000001), IFFF_RWBITS ($00000001), IFFF_FSEEK ($00000002), IFFF_RSEEK ($00000004), IFFF_RESERVED ($FFFF0000)
streamHook = the Hook structure containing a callback to handle low level I/O
Result: none

InitIFFasClip (Revision 2.0)
Description: initializes an IFFHandle as a clipboard stream
Library: iffparse.library
Offset: -$F0
Syntax: InitIFFasClip(iff)
C: VOID InitIFFasClip(struct IFFHandle *)
ML: InitIFFasClip(a0)
Arguments: iff = an IFFHandle structure
Result: none

InitIFFasDOS (Revision 2.0)
Description: initializes an IFFHandle as a DOS stream
Library: iffparse.library
Offset: -$EA
Syntax: InitIFFasDOS(iff)
C: InitIFFasDOS(struct IFFHandle *)
ML: InitIFFasDOS(a0)
Arguments: iff = an IFFHandle structure
Result: none

InitLayers (Obsolete under Revision 1.2)
Description: initializes a Layer_Info structure
Library: layers.library
Offset: -$1E
Syntax: InitLayers(li)
C: VOID InitLayers(struct Layer_Info *)
ML: InitLayers(a0)
Arguments: li = the LayerInfo structure to initialize
Result: none

InitMasks
Description: initializes a VSprite's BorderLine and CollMask masks
Library: graphics.library
Offset: -$7E
Syntax: InitMasks(vs)
C: VOID InitMasks(struct VSprite *)
ML: InitMasks(a0)
Arguments: vs = VSprite structure
Result: none

InitRastPort
Description: initializes a RasterPort structure
Library: graphics.library
Offset: -$C6
Syntax: InitRastPort(rastPort)

C: VOID InitRastPort(struct RastPort *rastPort)
ML: InitRastPort(a1)
Arguments: rastPort = RastPort structure to initialize
Result: none

InitRequester

Description: initializes a Requester structure
Library: intuition.library
Offset: -$8A
Syntax: InitRequester(requester)
C: VOID InitRequester(struct Requester *)
ML: InitRequester(a0)
Arguments: requester = Requester structure to initialize
Result: none

InitResident

Description: initializes a resident module
Library: exec.library
Offset: -$66
Syntax: object = InitResident(resident, segList)
C: APTR InitResident(struct Resident *, ULONG)
ML: d0 = InitResident(a1, d1)
Arguments: resident = a ROMTag
segList = seglist of the loaded object;
if loaded from disk, Libraries & Devices will cache this value for later return at close or expunge time—pass 0 for ROM modules
Result: object = return value from the init code (usually the library or device base); 0 for failure

InitSemaphore

Description: initializes a signal semaphore
Library: exec.library
Offset: -$22E
Syntax: InitSemaphore(signalSemaphore)
C: VOID InitSemaphore(struct SignalSemaphore *)
ML: InitSemaphore(a0)
Arguments: signalSemaphore = a signal semaphore structure;
all fields should be set to zero
Result: none

InitStruct

Description: initializes memory from a table
Library: exec.library
Offset: -$4E
Syntax: InitStruct(initTable, memory, size);

C: VOID InitStruct(struct InitStruct *, APTR, ULONG)
ML: InitStruct(a1, a2, d0)
Arguments: initTable = the table containing the commands and data with which to init the memory; must be word-aligned unless only byte initialization is done end table with "dc.b 0" or "dc.w 0"
memory = the memory to initialize; must be word-aligned if size is specified
size = the size of memory; used to clear the memory before initilization from initTable if 0, the memory is not cleared first
Result: none

InitTmpRas

Description: initializes chip RAM for use by area-fill, flood-fill, and text functions
Library: graphics.library
Offset: -$1D4
Syntax: InitTmpRas(tmpRas, buffer, size)
C: VOID InitTmpRas(struct TmpRas *, VOID *, ULONG)
ML: InitTmpRas(a0, a1, d0)
Arguments: tmpRas = pointer to the commands and data used to initialize chip memory
buffer = chip memory to be initialized
size = size of buffer
Result: none

InitView

Description: initializes a View structure
Library: graphics.library
Offset: -$168
Syntax: InitView(view)
C: VOID InitView(struct View *)
ML: InitView(a1)
Arguments: view = View structure to initialize
Result: none

InitVPort

Description: initializes a ViewPort structure
Library: graphics.library
Offset: -$CC
Syntax: InitVPort(viewPort)
C: VOID InitVPort(struct ViewPort *)
ML: InitVPort(a0)
Arguments: viewPort = ViewPort structure to initialize
Result: none

Input

Description: identifies the program's initial input filehandle
Library: dos.library
Offset: -$36
Syntax: fh = Input( )
C: BPTR Input(VOID)
ML: d0 = Input( )
Arguments: none
Result: fh = input filehandle

Insert

Description: inserts a node into a list
Library: exec.library
Offset: -$EA
Syntax: Insert(list, node, listNode)
C: VOID Insert(struct List *, struct Node *, struct Node *)
ML: Insert(a0, a1, a2)
Arguments: list = the target list header
node = the node to insert
listNode = the node after which to insert
Result: none

InsertCxObj (Revision 2.0)

Description: inserts a commodity object into a list at a specific point
Library: commodities.library
Offset: -$60
Syntax: InsertCxObj(headObj, cxobj, pred)
C: VOID InsertCxObj(CxObj *, CxObj *, CxObj *)
ML: InsertCxObj(a0, a1, a2)
Arguments: headObj = list of objects
cxobj = commodity object to add
pred = object after which cxobj should be inserted
Result: none

InstallClipRegion

Description: installs a clip region in a layer
Library: layers.library
Offset: -$AE
Syntax: oldclipregion = InstallClipRegion(layer, region)
C: struct Region *InstallClipRegion(struct Layer *, struct Region *)
ML: d0 = InstallClipRegion(a0, a1)
Arguments: layer = the layer
region = the region to install
Result: oldclipregion = the clip region previously installed; zero if none previously installed

InstallLayerHook (Revision 2.0)

Description: safely installs a new Layer->BackFill hook
Library: layers.library
Offset: -$C6
Syntax: oldhook = InstallLayerHook(layer, hook)
C: struct Hook *InstallLayerHook(struct Layer *, struct Hook *)
ML: d0 = InstallLayerHook(a0, a1)
Arguments: layer = the layer in which to install the new Backfill Hook
hook = the new layer callback Hook
Result: oldhook = the Layer->BackFill Hook previously active; zero if the default,
(Revision 3.0:) 1 if none previously active

InstallLayerInfoHook (Revision 3.0)

Description: installs a backfill hook for a non-layer
Library: layers.library
Offset: -$CC
Syntax: oldhook = InstallLayerInfoHook(li, hook)
C: struct Hook *InstallLayerInfoHook(struct Layer_Info *, struct Hook *)
ML: d0 = InstallLayerInfoHook(a0, a1)
Arguments: li = a LayerInfo structure
hook = the callback function to install
Result: oldhook = the backfill hook previously installed; zero if the default,
1 if none previously installed, -1 if unsuccessful

InternalLoadSeg (Revision 2.0)

Description: performs a low-level segment load
Library: dos.library
Offset: -$2F4
Syntax: seglist = InternalLoadSeg(fh, table, functionarray, stack)
C: BPTR InternalLoadSeg(BPTR, BPTR, APTR *, LONG *)
ML: d0 = InternalLoadSeg(d0, a0, a1, a2)
Arguments: fh = filehandle to load from
table = table used when loading overlays (otherwise ignored)
functionarray = array of functions to be used for reading, allocating, and freeing
Result: seglist = seglist loaded; zero if unsuccessful

InternalUnLoadSeg (Revision 2.0)

Description: unloads a seglist loaded with InternalLoadSeg( )
Library: dos.library
Offset: -$2FA
Syntax: success = InternalUnLoadSeg(seglist, FreeFunc)
C: BOOL InternalUnLoadSeg(BPTR, VOID (*)( )
ML: d0 = InternalUnLoadSeg(d1, a1)

Arguments: seglist = seglist to be unloaded
FreeFunc = function called to free memory
Result: success = zero if unsuccessful

IntuiTextLength

Description: returns the pixel-width of an IntuiText string
Library: intuition.library
Offset: -$14A
Syntax: length = IntuiTextLength(iText)
C: LONG IntuiTextLength(struct IntuiText *)
ML: d0 = IntuiTextLength(a0)
Arguments: iText = IntuiText structure containing string in question
Result: length = pixel-width of string

InvertKeyMap (Revision 2.0)

Description: translates an ANSI code to an input event
Library: commodities.library
Offset: -$AE
Syntax: success = InvertKeyMap(ansiCode, event, km)
C: BOOL InvertKeyMap(ULONG, struct InputEvent *, struct KeyMap *)
ML: d0 = InvertKeyMap(d0, a0, a1)
Arguments: ansiCode = ANSI code to convert
event = InputEvent structure to fill in
km = keymap to use for the translation
Result: success = zero if unsuccessful

IoErr

Description: gets additional error information from the system
Library: dos.library
Offset: -$84
Syntax: error = IoErr( )
C: LONG IoErr(VOID)
ML: d0 = IoErr( )
Arguments: none
Result: error = zero if successful; nonzero error-code if unsuccessful

IsAlNum (Revision 3.0)

Description: tests whether a character is alphanumeric
Library: locale.library
Offset: -$54
Syntax: state = IsAlNum(locale, character)
C: BOOL IsAlNum(struct Locale *, ULONG)
ML: d0 = IsAlNum(a0, d0)

Arguments: locale = the locale to use for the test
character = the character to test
Result: state = zero if character is not alphanumeric

IsAlpha (Revision 3.0)
Description: tests whether a character is alphabetical
Library: locale.library
Offset: -$5A
Syntax: state = IsAlpha(locale, character)
C: BOOL IsAlpha(struct Locale *, ULONG)
ML: d0 = IsAlpha(a0, d0)
Arguments: locale = the locale to use for the test
character = the character to test
Result: state = zero if character is not alphabetical

IsCntrl (Revision 3.0)
Description: tests whether a character is a control character
Library: locale.library
Offset: -$60
Syntax: state = IsCntrl(locale, character)
C: BOOL IsCntrl(struct Locale *, ULONG)
ML: d0 = IsCntrl(a0, d0)
Arguments: locale = the locale to use for the test
character = the character to test
Result: state = zero if character is not a control character

IsDigit (Revision 3.0)
Description: tests whether a character is a decimal digit
Library: locale.library
Offset: -$66
Syntax: state = IsDigit(locale, character)
C: BOOL IsDigit(struct Locale *, ULONG)
ML: d0 = IsDigit(a0, d0)
Arguments: locale = the locale to use for the test
character = the character to test
Result: state = zero if character is not a decimal digit

IsFileSystem (Revision 2.0)
Description: returns whether a DOS handler is a filesystem
Library: dos.library
Offset: -$2C4
Syntax: result = IsFileSystem(name)
C: BOOL IsFileSystem(STRPTR)
ML: d0 = IsFileSystem(d1)

Arguments: name = name of device in question (with :)
Result: result = zero if not a filesystem

IsGraph (Revision 3.0)

Description: tests whether a character is visible
Library: locale.library
Offset: -$6C
Syntax: state = IsGraph(locale, character)
C: BOOL IsGraph(struct Locale *, ULONG)
ML: d0 = IsGraph(a0, d0)
Arguments: locale = the locale to use for the test
character = the character to test
Result: state = zero if character is not visible

IsInteractive

Description: returns whether a file is "interactive"
Library: dos.library
Offset: -$D8
Syntax: status = IsInteractive(fh)
C: BOOL IsInteractive(BPTR)
ML: d0 = IsInteractive(d1)
Arguments: fh = filehandle of file in question
Result: status = zero if not interactive

IsLower (Revision 3.0)

Description: tests whether a character is lower case
Library: locale.library
Offset: -$72
Syntax: state = IsLower(locale, character)
C: BOOL IsLower(struct Locale *, ULONG)
ML: d0 = IsLower(a0, d0)
Arguments: locale = the locale to use for the test
character = the character to test
Result: state = zero if character is not lower case

IsPrint (Revision 3.0)

Description: tests whether a character is blank
Library: locale.library
Offset: -$78
Syntax: state = IsPrint(locale, character)
C: BOOL IsPrint(struct Locale *, ULONG)
ML: d0 = IsPrint(a0, d0)
Arguments: locale = the locale to use for the test
character = the character to test
Result: state = zero if character is not blank

IsPunct (Revision 3.0)

Description: tests whether a character is punctuation
Library: locale.library
Offset: -$7E
Syntax: state = IsPunct(locale, character)
C: BOOL IsPunct(struct Locale *, ULONG)
ML: d0 = IsPunct(a0, d0)
Arguments: locale = the locale to use for the test
character = the character to test
Result: state = zero if character is not punctuation

IsRexxMsg (Revision 2.0)

Description: tests whether a message is an ARexx message
Library: rexxsyslib.library
Offset: -$A8
Syntax: result = IsRexxMsg(msgptr)
C: BOOL IsRexxMsg(struct RexxMsg *)
ML: d0 = IsRexxMsg(a0)
Arguments: msgptr = the message to test
Result: result = nonzero if the message is an ARexx message

IsSpace (Revision 3.0)

Description: tests whether a character is white space
Library: locale.library
Offset: -$84
Syntax: state = IsSpace(locale, character)
C: BOOL IsSpace(struct Locale *, ULONG)
ML: d0 = IsSpace(a0, d0)
Arguments: locale = the locale to use for the test
character = the character to test
Result: state = zero if character is not white space

IsUpper (Revision 3.0)

Description: tests whether a character is upper case
Library: locale.library
Offset: -$8A
Syntax: state = IsUpper(locale, character)
C: BOOL IsUpper(struct Locale *, ULONG)
ML: d0 = IsUpper(a0, d0)
Arguments: locale = the locale to use for the test
character = the character to test
Result: state = zero if character is not upper case

IsXDigit (Revision 3.0)

Description: tests whether a character is a hexadecimal digit
Library: locale.library
Offset: -$90
Syntax: state = IsXDigit(locale, character)
C: BOOL IsXDigit(struct Locale *, ULONG)
ML: d0 = IsXDigit(a0, d0)
Arguments: locale = the locale to use for the test
character = the character to test
Result: state = zero if character is not a hexadecimal digit

ItemAddress

Description: returns the address of a specified MenuItem
Library: intuition.library
Offset: -$90
Syntax: item = ItemAddress(menuStrip, menuNumber)
C: struct MenuItem *ItemAddress(struct Menu *, UWORD)
ML: d0 = ItemAddress(a0, d0)
Arguments: menuStrip = first menu in the menu strip
menuNumber = the menu and item numbers
Result: item = the specified MenuItem; zero if menuNumber equals zero

LayoutMenuItemsA (Revision 2.0)

Description: positions menu items based on VisualInfo structure
Library: gadtools.library
Offset: -$3C
Syntax: success = LayoutMenuItemsA(menuitem, vi, tags)
C: BOOL LayoutMenuItemsA(struct MenuItem *, APTR, struct TagItem *)
ML: d0 = LayoutMenuItemsA(a0, a1, a2)
Arguments: menuitem = list of MenuItem structures
vi = pointer to private data returned by GetVisualInfoA( )
tags = tag list of attributes—GTMN_TextAttr ($80080031), GTMN_FrontPen ($80080032), GTMN_Menu($8008003C), GTMN_FullMenu ($8008003E), Revision 3.0: GTMN_Checkmark ($80080041), GTMN_AmigaKey ($80080042), GTMN_NewLookMenus ($80080043)
Result: success = zero if unsuccessful

LayoutMenusA (Revision 2.0)

Description: positions menus and menu items based on VisualInfo structure
Library: gadtools.library
Offset: -$42
Syntax: success = LayoutMenusA(menu, vi, tags)
C: BOOL LayoutMenus(struct Menu *, APTR, struct TagItem *)
ML: d0 = LayoutMenusA(a0, a1, a2)

Arguments: menu = menu returned by CreateMenusA( )
vi = pointer to private data returned by GetVisualInfoA( )
tags = tag list of attributes—GTMN_TextAttr ($80080031), GTMN_FrontPen ($80080032), GTMN_Menu($8008003C), GTMN_FullMenu ($8008003E), Revision 3.0: GTMN_Checkmark ($80080041), GTMN_AmigaKey ($80080042), GTMN_NewLookMenus ($80080043)
Result: success = zero if unsuccessful

LendMenus (Revision 3.0)

Description: lends a window's menus to another window
Library: intuition.library
Offset: -$324
Syntax: LendMenus(borrowerWindow, lenderWindow)
C: VOID LendMenus(struct Window *, struct Window *)
ML: LendMenus(a0, a1)
Arguments: borrowerWindow = window to adopt menus
lenderWindow = window whose menus are being shared; 0 to repossess lent menus
Result: none

LengthArgstring (Revision 2.0)

Description: returns the length of an argument string
Library: rexxsyslib.library
Offset: -$8A
Syntax: length = LengthArgstring(argstring)
C: ULONG LengthArgstring(UBYTE *)
ML: d0 = LengthArgstring(a0)
Arguments: argstring = an argument string
Result: length = length of argstring

LoadRGB32 (Revision 3.0)

Description: sets a ViewPort's palette using 32-bit color values
Library: graphics.library
Offset: -$372
Syntax: LoadRGB32(viewPort, table)
C: VOID LoadRGB32(struct ViewPort *, ULONG *)
ML: LoadRGB32(a0, a1)
Arguments: viewPort = ViewPort whose colors you wish to change
colors = series of records which describe which colors to modify
Result: none

LoadRGB4

Description: sets a ViewPort's palette using 12-bit color values
Library: graphics.library
Offset: -$C0

Syntax: LoadRGB4(viewPort, colorMap, count)
C: VOID LoadRGB4(struct ViewPort *, UWORD *, WORD)
ML: LoadRGB4(a0, a1, d0)
Arguments: viewPort = ViewPort whose colors you wish to change
colorMap = array of 12-bit color values; one word per color
count = number of colors to change
Result: none

LoadSeg

Description: scatterloads a program file into memory
Library: dos.library
Offset: -$96
Syntax: seglist = LoadSeg(name)
C: BPTR LoadSeg(STRPTR)
ML: d0 = LoadSeg(d1)
Arguments: name = program filename
Result: seglist = the seglist for the loaded program

LoadView

Description: assigns a new copper list to the current display
Library: graphics.library
Offset: -$DE
Syntax: LoadView(View)
C: VOID LoadView(struct View *)
ML: LoadView(a1)
Arguments: View = View structure containing new copper list
Result: none

LocalItemData (Revision 2.0)

Description: gets a pointer to user data for a local context item
Library: iffparse.library
Offset: -$C0
Syntax: data = LocalItemData(localItem)
C: APTR LocalItemData(struct LocalContextItem *)
ML: d0 = LocalItemData(a0)
Arguments: localItem = a local context item
Result: data = a pointer to the user data area; zero if localItem is zero

Lock

Description: locks a directory or file
Library: dos.library
Offset: -$54
Syntax: lock = Lock(name, accessMode)
C: BPTR Lock(STRPTR, LONG)
ML: d0 = Lock(d1, d2)

Arguments: name = name of directory or file
accessMode = type of access desired
ACCESS_READ ($FFFFFFFE), ACCESS_WRITE ($FFFFFFFF)
Result: lock = lock on directory or file; zero if unsuccessful

LockAmigaGuideBase (Revision 3.0)
Description: locks a client
Library: amigaguide.library
Offset: -$24
Syntax: key = LockAmigaGuideBase(handle)
C: LONG LockAmigaGuideBase(AMIGAGUIDECONTEXT)
ML: d0 = LockAmigaGuideBase(a0)
Arguments: handle = handle to an AmigaGuide system
Result: key = value to pass to UnlockAmigaGuideBase( )

LockDosList (Revision 2.0)
Description: locks the specified DosLists, waiting for it to become available, if necessary
Library: dos.library
Offset: -$28E
Syntax: dlist = LockDosList(flags)
C: struct DosList *LockDosList(ULONG)
ML: d0 = LockDosList(d1)
Arguments: flags = types of nodes you want to lock:
LDF_DEVICES ($00000004), LDF_VOLUMES ($00000008), LDF_ASSIGNS ($00000010), LDF_ENTRY ($00000020)
Result: dlist = pointer that you pass to NextDosEntry( ) to access DosList

LockIBase
Description: locks IntuitionBase to prevent changes
Library: intuition.library
Offset: -$19E
Syntax: lock = LockIBase(lockNumber)
C: ULONG LockIBase(ULONG)
ML: d0 = LockIBase(d0)
Arguments: lockNumber = desired Intuition lock; should always be zero
Result: lock = value to use to unlock IntuitionBase via UnlockIBase( )

LockLayer
Description: locks a layer to allow changes to ClipRects
Library: layers.library
Offset: -$60
Syntax: LockLayer(dummy, layer)
C: VOID LockLayer(LONG, struct Layer *)
ML: LockLayer(a0, a1)

Arguments: dummy = unused
layer = the layer to lock
Result: none

LockLayerInfo
Description: locks a LayerInfo structure
Library: layers.library
Offset: -$78
Syntax: LockLayerInfo(li)
C: VOID LockLayerInfo(struct Layer_Info *)
ML: LockLayerInfo(a0)
Arguments: li = the Layer_Info structure to lock
Result: none

LockLayerRom
Description: locks a Layer structure (replaces old LockLayer( ) function)
Library: graphics.library
Offset: -$1B0
Syntax: LockLayerRom(layer)
C: VOID LockLayerRom(struct Layer *)
ML: LockLayerRom(a5)
Arguments: layer = Layer structure to lock
Result: none

LockLayers
Description: locks all layers from graphics output
Library: layers.library
Offset: -$6C
Syntax: LockLayers(li)
C: VOID LockLayers(struct Layer_Info *)
ML: LockLayers(a0)
Arguments: li = the Layer_Info structure containing the list of layers to lock
Result: none

LockPubScreen (Revision 2.0)
Description: prevents a public screen from closing
Library: intuition.library
Offset: -$1FE
Syntax: screen = LockPubScreen(name)
C: struct Screen *LockPubScreen(UBYTE *)
ML: d0 = LockPubScreen(a0)
Arguments: name = name of public screen to lock; 0 for default screen
Result: screen = the locked screen; zero if unsuccessful

LockPubScreenList (Revision 2.0)

Description: prevents changes to the system list of public screens
Library: intuition.library
Offset: -$20A
Syntax: list = LockPubScreenList( )
C: struct List *LockPubScreenList(VOID)
ML: d0 = LockPubScreenList( )
Arguments: none
Result: list = the public screen list

LockRecord (Revision 2.0)

Description: locks a portion of a file
Library: dos.library
Offset: -$10E
Syntax: success = LockRecord(fh, offset, length, mode, timeout)
C: ULONG LockRecord(BPTR, ULONG, ULONG, ULONG, ULONG)
ML: d0 = LockRecord(d1, d2, d3, d4, d5)
Arguments: fh = filehandle of file containing record
offset = start position of record
length = length of record in bytes
mode = type of lock requester:
REC_EXCLUSIVE ($00000000), REC_EXCLUSIVE_IMMED ($00000001),
REC_SHARED ($00000002), REC_SHARED_IMMED ($00000003)
timeout = timeout interval in ticks (50ths of a second)
Result: success = zero if unsuccessful

LockRecords (Revision 2.0)

Description: locks a series of records
Library: dos.library
Offset: -$114
Syntax: success = LockRecords(recordArray, timeout)
C: BOOL LockRecords(struct RecordLock *, ULONG)
ML: d0 = LockRecords(d1, d2)
Arguments: recordArray = list of records to be locked
timeout = timeout interval
Result: success = zero if unsuccessful

LockRexxBase (Revision 2.0)

Description: obtains a semaphore lock on the RexxBase structure
Library: rexxsyslib.library
Offset: -$1C2
Syntax: LockRexxBase(resource)
C: VOID LockRexxBase(ULONG)
ML: LockRexxBase(d0)

Arguments: resource = which resources to lock; must be 0 (all resources)
Result: none

MakeClass (Revision 2.0)
Description: creates and initializes a boopsi class object
Library: intuition.library
Offset: -$2A6
Syntax: iclass = MakeClass(classID, superClassID, superClassPtr, instanceSize, flags)
C: struct IClass *MakeClass(UBYTE *, UBYTE *, struct IClass *, UWORD, ULONG)
ML: d0 = MakeClass(a0, a1, a2, d0, d1)
Arguments: classID = the name/ID string for public classes; 0 for private classes
superClassID = name/ID of your new class's superclass; 0 if superclass is a private class
superClassPtr = a private superclass; used only if SuperClassID is 0
instanceSize = size of the instance data required by the class's objects
flags = must be zero
Result: iclass = the resulting class; zero if unsuccessful

MakeDosEntry (Revision 2.0)
Description: creates a DosList structure
Library: dos.library
Offset: -$2B8
Syntax: newdlist = MakeDosEntry(name, type)
C: struct DosList *MakeDosEntry(STRPTR, LONG)
ML: d0 = MakeDosEntry(d1, d2)
Arguments: name = name for the device/volume/assign node
type = type of node:
LDF_DEVICES ($00000004), LDF_VOLUMES ($00000008), LDF_ASSIGNS ($00000010), LDF_ENTRY ($00000020)
Result: newdlist = the new device entry; zero if unsuccessful

MakeDosNode
Description: constructs the DOS data structures that a disk needs
Library: expansion.library
Offset: -$90
Syntax: deviceNode = MakeDosNode(parameterPkt)
C: struct DeviceNode * MakeDosNode(VOID *)
ML: d0 = MakeDosNode(a0)
Arguments: parameterPkt = a longword array containing initialization data
Result: deviceNode = pointer to initialized device node structure; zero if unsuccessful

MakeFunctions
Description: constructs a function jump table
Library: exec.library

Offset: -$5A
Syntax: tableSize = MakeFunctions(target, functionArray, funcDispBase)
C: ULONG MakeFunctions(APTR, APTR, APTR)
ML: d0 = MakeFunctions(a0, a1, a2)
Arguments: target = the address of the end of the table
functionArray = -1-terminated array of function pointers or word displacements; if funcDispBase is 0, this array contains absolute addresses, otherwise it contains word displacements relative to funcDispBase
funcDispBase = base address of function displacements; 0 indicates absolute addresses
Result: tableSize = size of the new table in bytes

MakeLibrary

Description: constructs a library
Library: exec.library
Offset: -$54
Syntax: library = MakeLibrary(vectors, structure, init, dSize, segList)
C: struct Library *MakeLibrary(APTR, struct InitStruct *, APTR, ULONG, BPTR)
ML: d0 = MakeLibrary(a0, a1, a2, d0, d1)
Arguments: vectors = -1-terminated array of function pointers or function displacements; if the first entry is -1, the entries are displacements, otherwise full addresses
structure = an InitStruct data region; may be 0
init = a initialization routine to be called before adding the library to the system; may be 0
dsize = size of the library data area, including the standard library node data
segList = an AmigaDOS seglist
Result: library = the reference address of the library; zero if out of memory

MakeLink (Revision 2.0)

Description: creates a file system link
Library: dos.library
Offset: -$1BC
Syntax: success = MakeLink(name, dest, type)
C: BOOL MakeLink(STRPTR, LONG, LONG)
ML: d0 = MakeLink(d1, d2, d3)
Arguments: name = name of the link to create
dest = path string or BPTR lock
type = what kind of link; zero for hardlinks, nonzero for softlinks
Result: success = zero if unsuccessful

MakeScreen

Description: executes an Intuition-integrated MakeVPort( ) of a custom screen
Library: intuition.library
Offset: -$17A
Syntax: MakeScreen(screen)

C: VOID MakeScreen(struct Screen *)
ML: MakeScreen(a0)
Arguments: screen = custom screen
Result: none

MakeVPort

Description: constructs a copper list for a ViewPort
Library: graphics.library
Offset: -$D8
Syntax: error = MakeVPort(view, viewport)
C: ULONG MakeVPort(struct View *, struct ViewPort *)
ML: d0 = MakeVPort(a0, a1)
Arguments: view = View structure
viewport = ViewPort structure
Result: error = zero if successful (valid under Revision 2.0 and 3.0 only)

MapANSI (Revision 2.0)

Description: translates an ANSI string to keycodes
Library: keymap.library
Offset: -$30
Syntax: actual = MapANSI(string, count, buffer, length, keyMap)
C: LONG MapANSI(STRPTR, LONG, STRPTR, LONG, struct KeyMap *)
ML: d0 = MapANSI(a0, d0, a1, d1, a2)
Arguments: string = ANSI string to convert
count = number of characters in the string
buffer = memory buffer to receive converted string
length = number of bytes in the buffer divided by two (the number code/qualifier pairs it will hold)
keyMap = KeyMap structure; 0 indicates the default key map
Result: actual = the number of code/qualifier pairs put into the buffer;
zero if the string was not translatable,
-1 if a buffer overflow occured,
-2 if an internal error occured

MapRawKey (Revision 2.0)

Description: decodes a single IECLASS_RAWKEY input event to an ANSI string
Library: keymap.library
Offset: -$2A
Syntax: actual = MapRawKey(event, buffer, length, keyMap)
C: WORD MapRawKey(struct InputEvent *, STRPTR, WORD, struct Keymap *)
ML: d0 = MapRawKey(a0, a1, d1, a2)

Arguments: event = the InputEvent structure to decode
buffer = memory buffer to receive the string
length = number of bytes in the buffer
keyMap = KeyMap structure; 0 indicates the default key map
Result: actual = the number of characters put into the buffer; -1 if a buffer overflow occurred

MapTags (Revision 2.0)
Description: remaps ti_Tag values in a tag list
Library: utility.library
Offset: -$3C
Syntax: MapTags(tags, mapList, mapType)
C: VOID MapTags(struct TagItem *, struct TagItem *, ULONG)
ML: MapTags(a0, a1, d0)
Arguments: tags = tag list to remap
mapList = tag list containing pairs of tag values;
the first in each pair to be replaced with the second
mapType = mapping operation
MAP_REMOVE_NOT_FOUND ($00000000), MAP_KEEP_NOT_FOUND ($00000001)
Result: none

MatchEnd (Revision 2.0)
Description: frees storage allocated by MatchFirst( ) or MatchNext( )
Library: dos.library
Offset: -$342
Syntax: MatchEnd(anchorpath)
C: VOID MatchEnd(struct AnchorPath *)
ML: MatchEnd(d1)
Arguments: anchorpath = anchorpath used in MatchFirst( ) or MatchNext( )
Result: none

MatchFirst (Revision 2.0)
Description: finds the first file or directory that matches a pattern
Library: dos.library
Offset: -$336
Syntax: error = MatchFirst(pat, anchorpath)
C: LONG MatchFirst(STRPTR, struct AnchorPath *)
ML: d0 = MatchFirst(d1, d2)
Arguments: pat = pattern to match
anchorpath = placeholder for search
Result: error = zero if successful; nonzero error-code if unsuccessful

MatchIX (Revision 2.0)
Description: matches an input event to an initialized input expression
Library: commodities.library

Offset: -$CC
Syntax: success = MatchIX(event, ix)
C: BOOL MatchIX(struct InputEvent *, IX *)
ML: d0 = MatchIX(a0, a1)
Arguments: event = input event to match
ix = input expression for the comparison
Result: success = zero if event and ix don't match

MatchNext (Revision 2.0)

Description: finds the next file or directory that matches a pattern
Library: dos.library
Offset: -$33C
Syntax: error = MatchNext(anchorpath)
C: LONG MatchNext(struct AnchorPath *)
ML: d0 = MatchNext(d1)
Arguments: anchorpath = place holder for search
Result: error = zero if successful; nonzero error-code if unsuccessful

MatchPattern (Revision 2.0)

Description: checks to see if a string matches a specified pattern
Library: dos.library
Offset: -$34E
Syntax: match = MatchPattern(pat, str)
C: BOOL MatchPattern(STRPTR, STRPTR)
ML: d0 = MatchPattern(d1, d2)
Arguments: pat = pattern string as returned by ParsePattern( )
str = string to check against given pattern
Result: match = nonzero if string matches pattern

MatchPatternNoCase (Revision 2.0)

Description: checks to see if a string matches a specified pattern, ignoring case
Library: dos.library
Offset: -$3C6
Syntax: match = MatchPatternNoCase(pat, str)
C: BOOL MatchPatternNoCase(STRPTR, STRPTR)
ML: d0 = MatchPatternNoCase(d1, d2)
Arguments: pat = pattern string as returned by ParsePattern( )
str = string to check against given pattern
Result: match = nonzero if string matches pattern

MatchToolValue

Description: checks a tool type variable for a particular value
Library: icon.library
Offset: -$66
Syntax: result = MatchToolValue(typeString, value)

C: BOOL MatchToolValue(char *, char *)
ML: d0 = MatchToolValue(a0, a1)
Arguments: typeString = a ToolType value returned by FindToolType( )
value = value to check for
Result: result = nonzero if value matches string; zero if value doesn't match

MaxCli (Revision 2.0)
Description: returns the highest CLI process number possibly in use
Library: dos.library
Offset: -$228
Syntax: number = MaxCli( )
C: LONG MaxCli(VOID)
ML: d0 = MaxCli( )
Arguments: none
Result: number = the highest CLI process number possibly in use

ModeNotAvailable (Revision 2.0)
Description: checks to see if a DisplayID is available
Library: graphics.library
Offset: -$31E
Syntax: error = ModeNotAvailable(modeID)
C: ULONG = ModeNotAvailable(ULONG)
ML: d0 = ModeNotAvailable(d0)
Arguments: modeID = a 32-bit DisplayInfoRecord identifier
Result: error = DI_AVAIL_NOCHIPS ($00000001), DI_AVAIL_NOMONITOR ($00000002), or DI_AVAIL_NOTWITHGENLOCK ($00000004); zero if DisplayID is available

ModifyIDCMP
Description: modifies the state of a window's IDCMP
Library: intuition.library
Offset: -$96
Syntax: success = ModifyIDCMP(window, idcmpFlags)
C: BOOL ModifyIDCMP(struct Window *, ULONG)
ML: d0 = ModifyIDCMP(a0, d0)
Arguments: window = window to modify
idcmpFlags = bit flags describing the desired state of the IDCMP
Result: success = zero if unsuccessful (valid under Revision 2.0 only)

ModifyProp
Description: modifies the current state of a proportional gadget (outdated by NewModifyProp( ) function)
Library: intuition.library
Offset: -$9C

Syntax: ModifyProp(propGadget, window, requester, flags, horizPot, vertPot, horizBody, vertBody)
C: VOID ModifyProp(struct Gadget *, struct Window *, struct Requester *, UWORD, UWORD, UWORD, UWORD, UWORD)
ML: ModifyProp(a0, a1, a2, d0, d1, d2, d3, d4)
Arguments: propGadget = proportional gadget to modify
window = window containing gadget
requester = requester containing gadget; may be 0
flags = new flag values for gadget—FREEHORIZ ($0002), FREEVERT ($0004), PROPBORDERLESS ($0008), PROPNEWLOOK ($0010), KNOBHIT ($0100)
horizPot = new horizontal pot value
vertPot = new vertical pot value
horizBody = new horizontal body value
vertBody = new vertical body value
Result: none

Move

Description: moves graphics pen
Library: graphics.library
Offset: -$F0
Syntax: Move(rastPort, x, y)
C: VOID Move(struct RastPort *, WORD, WORD)
ML: Move(a1, d0, d1)
Arguments: rastPort = RastPort structure
x, y = new pen position in the RastPort
Result: none

MoveLayer

Description: moves a layer to a new position in its BitMap
Library: layers.library
Offset: -$3C
Syntax: success = MoveLayer(dummy, layer, dx, dy)
C: LONG MoveLayer(LONG, struct Layer *, LONG, LONG)
ML: d0 = MoveLayer(a0, a1, d0, d1)
Arguments: dummy = unused
layer = pointer to the layer; must not be a backdrop layer
dx = amount to change the current x position
dy = amount to change the current y position
Result: success = zero if unsuccessful

MoveLayerInFrontOf

Description: moves a layer in front of another layer
Library: layers.library
Offset: -$A8
Syntax: success = MoveLayerInFrontOf(layertomove, targetlayer)

C: LONG MoveLayerInFrontOf(struct Layer *, struct Layer *)
ML: d0 = MoveLayerInFrontOf(a0, a1)
Arguments: layertomove = the layer to move
targetlayer = the layer in front of which to move
Result: success = zero if unsuccessful

MoveScreen

Description: moves a screen
Library: intuition.library
Offset: -$A2
Syntax: MoveScreen(screen, deltaX, deltaY)
C: VOID MoveScreen(struct Screen *, WORD, WORD)
ML: MoveScreen(a0, d0, d1)
Arguments: screen = screen to move
deltaX = offset by which to move screen horizontally; ignored
deltaY = offset by which to move screen vertically
Result: none

MoveSizeLayer (Revision 2.0)

Description: positions and sizes a layer
Library: layers.library
Offset: -$B4
Syntax: success = MoveSizeLayer(layer, dx, dy, dw, dh)
C: LONG MoveSizeLayer(struct Layer *, LONG, LONG, LONG, LONG)
ML: d0 = MoveSizeLayer(a0, d0, d1, d2, d3)
Arguments: dummy = unused
layer = the layer to move and size; must not be a backdrop layer
dx = amount to change the current x position
dy = amount to change the current y position
dw = amount to change the current width
dh = amount to change the current height
Result: success = zero if unsuccessful

MoveSprite

Description: moves a sprite
Library: graphics.library
Offset: -$1AA
Syntax: MoveSprite(viewPort, sprite, x, y)
C: VOID MoveSprite(struct ViewPort *, struct SimpleSprite *, WORD, WORD)
ML: MoveSprite(a0, a1, d0, d1)
Arguments: viewPort = ViewPort containing sprite; 0 to position sprite relative to current View
sprite = SimpleSprite structure
x, y = new sprite position
Result: none

MoveWindow

Description: moves a window
Library: intuition.library
Offset: -$A8
Syntax: MoveWindow(window, deltaX, deltaY)
C: VOID MoveWindow(struct Window *, WORD, WORD)
ML: MoveWindow(a0, d0, d1)
Arguments: window = window to move
deltaX = offset by which to move window horizontally
deltaY = offset by which to move window vertically
Result: none

MoveWindowInFrontOf (Revision 2.0)

Description: changes the relative depth of a window
Library: intuition.library
Offset: -$1E0
Syntax: MoveWindowInFrontOf(window, behindWindow)
C: VOID MoveWindowInFrontOf(struct Window *, struct Window *)
ML: MoveWindowInFrontOf(a0, a1)
Arguments: window = window to move up front
behindWindow = window to be obscured
Result: none

MrgCop

Description: merges copper instructions into copper list
Library: graphics.library
Offset: -$D2
Syntax: error = MrgCop(View)
C: ULONG MrgCop(struct View *)
ML: d0 = MrgCop(a1)
Arguments: View = View structure whose copper instructions are to be merged
Result: error = zero if successful (valid under Revision 2.0 and 3.0 only)

NamedObjectName (Revision 3.0)

Description: obtains the name of an object
Library: utility.library
Offset: -$FC
Syntax: name = NamedObjectName(object)
C: char *NamedObjectName(struct NamedObject *)
ML: d0 = NamedObjectName(a0)
Arguments: object = the object
Result: name = the name string

NameFromFH (Revision 2.0)

Description: gets the name of an open file from a filehandle
Library: dos.library
Offset: -$198
Syntax: success = NameFromFH(fh, buffer, len)
C: BOOL NameFromFH(BPTR, STRPTR, LONG)
ML: d0 = NameFromFH(d1, d2, d3)
Arguments: fh = filehandle of file to be examined.
buffer = buffer to receive name
len = size of buffer
Result: success = zero if unsuccessful

NameFromLock (Revision 2.0)

Description: gets the name of a file from a file lock
Library: dos.library
Offset: -$192
/Syntax: success = NameFromLock(lock, buffer, len)
C: BOOL NameFromLock(BPTR, STRPTR, LONG)
ML: d0 = NameFromLock(d1, d2, d3)
Arguments: lock = lock on file to be examined
buffer = buffer to receive name
len = size of buffer
Result: success = zero if unsuccessful

NewDTObjectA (Revision 3.0)

Description: allocates a DataType object
Library: datatypes.library
Offset: -$30
Syntax: object = NewDTObjectA(name, tags)
C: Object *NewDTObjectA(APTR, struct TagItem *)
ML: d0 = NewDTObjectA(d0, a0)
Arguments: name = the name of data source; usually the name of an existing file
tags = tag list specifying allocate-time attributes
DTA_SourceType ($80001065), DTA_Handle ($80001066), DTA_DataType ($80001067), DTA_GroupID ($8000101F), GA_Left ($80030001), GA_RelRight ($80030002), GA_Top ($80030003), GA_RelBottom ($80030004), GA_Width ($80030005), GA_RelWidth ($80030006), GA_Height ($80030007), GA_RelHeight ($80030008), GA_ID ($80030010), GA_UserData ($80030011), GA_Previous ($8003001F)
Result: object = a boopsi object

NewFontContents (Revision 2.0)

Description: creates FontContents entries for a font
Library: diskfont.library
Offset: -$2A

Syntax: fontContentsHeader = NewFontContents(fontsLock,fontName)
C: struct FontContentsHeader *NewFontContents(BPTR, char *)
ML: d0 = NewFontContents(a0, a1)
Arguments: fontsLock = DOS lock on the directory containing the font
fontName = font name (including the ".font" suffix)
Result: fontContentsHeader = a FontContentsHeader structure

NewLayerInfo

Description: allocates and initializes a Layer_Info structure
Library: layers.library
Offset: -$90
Syntax: result = NewLayerInfo( )
C: struct Layer_Info *NewLayerInfo(VOID)
ML: d0 = NewLayerInfo( )
Arguments: none
Result: result = an initialized Layer_Info structure; zero if unsuccessful

NewLoadSeg (Revision 2.0)

Description: an improved version of LoadSeg( )
Library: dos.library
Offset: -$300
Syntax: seglist = NewLoadSeg(file, tags)
C: BPTR NewLoadSegTags(STRPTR, struct TagItem *)
ML: d0 = NewLoadSeg(d1, d2)
Arguments: file = filename of file to load
tags = tag list of attributes—none are currently defined
Result: seglist = seglist loaded; zero if unsuccessful

NewModifyProp

Description: selectively refreshes a proportional gadget
Library: intuition.library
Offset: -$1D4
Syntax: NewModifyProp(propGadget, window, requester, flags, horizPot, vertPot, horizBody, vertBody, numGad)
C: VOID NewModifyProp(struct Gadget *, struct Window *, struct Requester *, UWORD, UWORD, UWORD, UWORD, UWORD, WORD)
ML: NewModifyProp(a0, a1, a2, d0, d1, d2, d3, d4, d5)
Arguments: propGadget = proportional gadget to modify
window = window containing gadget
requester = requester containing gadget; may be 0
flags = new flag values for gadget
FREEHORIZ ($0002), FREEVERT ($0004), PROPBORDERLESS ($0008), PROPNEWLOOK ($0010), KNOBHIT ($0100)
horizPot = new horizontal pot value
vertPot = new vertical pot value

horizBody = new horizontal body value
vertBody = new vertical body value
numGad = number of gadgets to refresh; -1 to refresh to the end of the gadget list
Result: none

NewObject (Revision 2.0)
Description: create an object from a class
Library: intuition.library
Offset: -$27C
Syntax: object = NewObjectA(class, classID, tags)
C: APTR NewObject(struct IClass *, UBYTE *, struct TagItem *)
ML: d0 = NewObjectA(a0, a1, a2)
Arguments: class = a boopsi class obtained from MakeClass( )
classID = the name/ID string of a public class; used only if class is zero
tags = tag list of attributes—see <intuition/classusr.h> for description
Result: object = a new boopsi object

NewRegion
Description: creates an empty Region structure
Library: graphics.library
Offset: -$204
Syntax: region = NewRegion( )
C: struct Region *NewRegion( )
ML: d0 = NewRegion( )
Arguments: none
Result: region = newly created Region structure

NewScaledDiskFont (Revision 2.0)
Description: creates a scaled DiskFont
Library: diskfont.library
Offset: -$36
Syntax: header = NewScaledDiskFont(srcFont, destTextAttr)
C: struct DiskFontHeader *NewScaledDiskFont(struct TextFont *, struct TTextAttr *)
ML: d0 = NewScaledDiskFont(a0, a1)
Arguments: srcFont = TextFont structure returned from OpenFont( ) or OpenDiskFont( )
destTextAttr = desired attributes for the new font; may be a TextAttr
structure or a TTextAttr structure
Result: header = a DiskFontHeader structure for the new font

NextDisplayInfo (Revision 2.0)
Description: gets the next DisplayInfo record parameters
Library: graphics.library
Offset: -$2DC
Syntax: nextID = NextDisplayInfo(lastID)
C: ULONG NextDisplayInfo(ULONG)

ML: d0 = NextDisplayInfo(d0)
Arguments: lastID = the DisplayInfo identifier returned by a previous call to this function; -1 to start with the first record
Result: nextID = subsequent DisplayInfo identifier; -1 if there are no more records

NextDosEntry (Revision 2.0)

Description: gets the next DosList entry
Library: dos.library
Offset: -$2B2
Syntax: newdlist = NextDosEntry(dlist, flags)
C: struct DosList *NextDosEntry(struct DosList *, ULONG)
ML: d0 = NextDosEntry(d1, d2)
Arguments: dlist = the current device entry
flags = types of entries to look for—LDF_DEVICES ($00000004), LDF_VOLUMES ($00000008), LDF_ASSIGNS ($00000010), LDF_ENTRY ($00000020)
Result: newdlist = the next DosList entry; zero if end of DosList

NextObject (Revision 2.0)

Description: iterates through the objects on an Exec list
Library: intuition.library
Offset: -$29A
Syntax: object = NextObject(objectPtrPtr)
C: APTR NextObject(APTR)
ML: d0 = NextObject(a0)
Arguments: objectPtrPtr = initially, the address of the lh_Head field of the list; for subsequent objects, pass the address of that pointer until 0 is returned
Result: object = next object; zero if there are no more objects

NextPubScreen (Revision 2.0)

Description: returns the next public screen in the list of screens
Library: intuition.library
Offset: -$216
Syntax: buff = NextPubScreen(screen, nameBuff)
C: UBYTE *NextPubScreen(struct Screen *, UBYTE *)
ML: d0 = NextPubScreen(a0, a1)
Arguments: screen = current screen; 0 to find the first public screen
nameBuff = buffer to receive the name; should be 140 characters
Result: buff = the nameBuff pointer; zero if there are no public screens

NextTagItem (Revision 2.0)

Description: iterates through a tag list
Library: utility.library
Offset: -$30
Syntax: tag = NextTagItem(tagItemPtr)

C: struct TagItem *NextTagItem(struct TagItem **)
ML: d0 = NextTagItem(a0)
Arguments: tagItemPtr = doubly-indirect reference to a TagItem structure used to maintain position
Result: nextTag = next valid tagitem in the tag list

ObtainBestPenA (Revision 3.0)

Description: searches for the closest color match
Library: graphics.library
Offset: -$348
Syntax: color = ObtainBestPenA(colorMap, red, green, blue, tags)
C: LONG = ObtainBestPenA(struct ColorMap *, ULONG, ULONG, ULONG, struct TagItem *)
ML: d0 = ObtainBestPenA(a0, d1, d2, d3, a1)
Arguments: colorMap = colormap
red, green, blue = color attributes
tags = tag list of attributes—OBP_Precision ($84000000), OBP_FailIfBad ($84000001)
Result: color = correct pen value; -1 if sharable palette entries are available

ObtainConfigBinding

Description: tries to lock ConfigDev list and get permission to bind new drivers
Library: expansion.library
Offset: -$78
Syntax: ObtainConfigBinding( )
C: VOID ObtainConfigBinding(VOID)
ML: ObtainConfigBinding( )
Arguments: none
Result: none

ObtainDataTypeA (Revision 3.0)

Description: returns the DataType of a handle
Library: datatypes.library
Offset: -$24
Syntax: dtn = ObtainDataTypeA(type, handle, tags)
C: struct DataType *ObtainDataTypeA(ULONG, APTR, struct TagItem *)
ML: d0 = ObtainDataTypeA(d0, a0, a1)
Arguments: type = type of handle:
DTST_RAM ($00000001), DTST_FILE ($00000002), DTST_CLIPBOARD ($00000003), DTST_HOTLINK ($00000004)
handle = handle to examine; for DTST_FILE, handle should be a BPTR lock, for DTST_CLIPBOARD, handle should be a pointer to an IFFHandle structure
tags = tag list specifying addtional attributes—none are currently defined
Result: dtn = a DataType structure; zero if unsuccessful

ObtainGIRPort (Revision 2.0)

Description: sets up a RastPort for a custom gadget
Library: intuition.library
Offset: -$22E
Syntax: rastPort = ObtainGIRPort(gInfo)
C: struct RastPort *ObtainGIRPort(struct GadgetInfo *)
ML: d0 = ObtainGIRPort(a0)
Arguments: gInfo = GadgetInfo structure
Result: rastPort = RastPort structure to be used for gadget rendering; zero if no rendering will be done

ObtainInfoA (Revision 3.0)

Description: gets font metrics
Library: bullet.library
Offset: -$30
Syntax: error = ObtainInfoA(handle, tags)
C: ULONG ObtainInfo(struct GlyphEngine *, struct TagItem *)
ML: d0 = ObtainInfoA(a0, a1)
Arguments: handle = handle returned by OpenEngine( )
tags = tag list specifying information requested:
see <diskfont/diskfonttag.h> for description
Result: error = zero if successful; nonzero error-code if unsuccessful

ObtainPen (Revision 3.0)

Description: obtains a free palette entry
Library: graphics.library
Offset: -$3BA
Syntax: pen = ObtainPen(colorMap, pen, red, green, blue, flags)
C: LONG ObtainPen(struct ColorMap *, ULONG, ULONG, ULONG, ULONG, ULONG)
ML: d0 = ObtainPen(a0, d0, d1, d2, d3, d4)
Arguments: colorMap = ColorMap structure
pen = desired entry number; -1 if you don't care
red, green, blue = color attributes
flags = 0 if you can share this color with other tasks, PEN_EXCLUSIVE ($00000001) to restrict access
Result: pen = free palette entry; -1 if unsuccessful

ObtainSemaphore

Description: obtains exclusive access to a semaphore
Library: exec.library
Offset: -$234
Syntax: ObtainSemaphore(signalSemaphore)
C: VOID ObtainSemaphore(struct SignalSemaphore *)
ML: ObtainSemaphore(a0)

Arguments: signalSemaphore = an initialized signal semaphore structure
Result: none

ObtainSemaphoreList
Description: obtains a list of semaphores
Library: exec.library
Offset: -$246
Syntax: ObtainSemaphoreList(list)
C: VOID ObtainSemaphoreList(struct List *)
ML: ObtainSemaphoreList(a0)
Arguments: list = a list of signal semaphores
Result: none

ObtainSemaphoreShared (Revision 2.0)
Description: obtains access to a shared semaphore
Library: exec.library
Offset: -$2A6
Syntax: ObtainSemaphoreShared(signalSemaphore)
C: VOID ObtainSemaphoreShared(struct SignalSemaphore *)
ML: ObtainSemaphoreShared(a0)
Arguments: signalSemaphore = an initialized signal semaphore structure
Result: none

OffGadget
Description: disables a gadget
Library: intuition.library
Offset: -$AE
Syntax: OffGadget(gadget, window, requester)
C: VOID OffGadget(struct Gadget *, struct Window *, struct Requester *)
ML: OffGadget(a0, a1, a2)
Arguments: Gadget = gadget to disable
window = window containing the gadget
requester = requester containing the gadget; may be 0
Result: none

OffMenu
Description: disables a menu or menu item
Library: intuition.library
Offset: -$B4
Syntax: OffMenu(window, menuNumber)
C: VOID OffMenu(struct Window *, UWORD)
ML: OffMenu(a0, d0)

Arguments: window = window containing the menu
menuNumber = the packed menu or menu item number—see <intuition/intuition.h> for description
Result: none

OldOpenLibrary

Description: provides support for an obsolete OpenLibrary
Library: exec.library
Offset: -$198
Syntax: library = OldOpenLibrary(libName)
C: struct Library *OldOpenLibrary(APTR)
ML: d0 = OldOpenLibrary(a1)
Arguments: libName = the name of the library to open
Result: library = the library base address; zero if unsuccessful

OnGadget

Description: enables a gadget
Library: intuition.library
Offset: -$BA
Syntax: OnGadget(gadget, window, requester)
C: VOID OnGadget(struct Gadget *, struct Window *, struct Requester *)
ML: OnGadget(a0, a1, a2)
Arguments: gadget = gadget to enable
window = window containing the gadget
requester = requester containing the gadget; may be 0
Result: none

OnMenu

Description: enables a menu or menu item
Library: intuition.library
Offset: -$C0
Syntax: OnMenu(window, menuNumber)
C: VOID OnMenu(struct Window *, UWORD)
ML: OnMenu(a0, d0)
Arguments: window = window containing the menu
menuNumber = the packed menu or menu item number—see <intuition/intuition.h> for description
Result: none

Open

Description: opens a file for input or output
Library: dos.library
Offset: -$1E
Syntax: fh = Open(name, accessMode)
C: BPTR Open(STRPTR, LONG)

ML: d0 = Open(d1, d2)
Arguments: name = filename of file to open
accessMode = type of file access desired
MODE_READWRITE ($000003EC), MODE_OLDFILE ($000003ED),
MODE_NEWFILE ($000003EE)
Result: fh = filehandle of open file; zero if unsuccessful

OpenAmigaGuideA (Revision 3.0)
Description: opens a synchronous database
Library: amigaguide.library
Offset: -$36
Syntax: handle = OpenAmigaGuideA(newag, tags)
C: AMIGAGUIDECONTEXT OpenAmigaGuideA(struct NewAmigaGuide *, struct TagItem *)
ML: d0 = OpenAmigaGuideA(a0, a1)
Arguments: newag = structure containing initialization data
tags = tag list of attributes—AGA_HelpGroup ($80000005)
Result: handle = handle to an AmigaGuide system

OpenAmigaGuideAsyncA (Revision 3.0)
Description: opens an asynchronous database
Library: amigaguide.library
Offset: -$3C
Syntax: handle = OpenAmigaGuideAsyncA(newag, tags)
C: AMIGAGUIDECONTEXT OpenAmigaGuideAsyncA(struct NewAmigaGuide *, struct TagItem *)
ML: d0 = OpenAmigaGuideA(a0, d0)
Arguments: newag = structure containing initialization data
tags = tag list of attributes—AGA_HelpGroup ($80000005)
Result: handle = handle to an AmigaGuide system

OpenCatalog (Revision 3.0)
Description: opens a message catalog
Library: locale.library
Offset: -$96
Syntax: catalog = OpenCatalogA(locale, name, tags)
C: struct Catalog *OpenCatalog(struct Locale *, STRPTR, struct TagItem *)
ML: d0 = OpenCatalogA(a0, a1, a2)
Arguments: locale = the locale for which the catalog should be opened; 0 indicates the system default
name = the catalog to open
tags = tag list of attributes:
OC_BuiltInLanguage ($00090001), OC_BuiltInCodeSet ($00090002),
OC_Version ($00090003), OC_Language ($00090004)
Result: catalog = a message catalog; zero in extenuating circumstances, call IoErr( ) to test for error

OpenClipboard (Revision 2.0)
Description: creates a handle on a clipboard unit
Library: iffparse.library
Offset: -$FC
Syntax: ch = OpenClipboard(unitNumber)
C: struct ClipboardHandle *OpenClipboard(LONG)
ML: d0 = OpenClipboard(d0)
Arguments: unitNumber = clipboard unit number usually PRIMARY_CLIP ($00000000)
Result: ch = a ClipboardHandle structure; zero if unsuccessful

OpenDevice
Description: gains access to a device
Library: exec.library
Offset: -$1BC
Syntax: error = OpenDevice(devName, unitNumber, iORequest, flags)
C: BYTE OpenDevice(STRPTR, ULONG, struct IORequest *, ULONG)
ML: d0 = OpenDevice(a0, d0, a1, d1)
Arguments: devName = name of the device requested
Result: error = zero if successful; a sign-extended copy of the io_Error field of the IORequest if unsuccessful

OpenDiskFont
Description: loads a disk font
Library: diskfont.library
Offset: -$1E
Syntax: font = OpenDiskFont(textAttr)
C: struct TextFont *OpenDiskFont(struct TextAttr *textAttr)
ML: d0 = OpenDiskFont(a0)
Arguments: textAttr = the font to open
Result: font = a TextFont structure; zero if unsuccessful

OpenEngine (Revision 3.0)
Description: acquires an engine handle
Library: bullet.library
Offset: -$1E
Syntax: handle = OpenEngine( )
C: struct GlyphEngine *OpenEngine(VOID)
ML: d0 = OpenEngine( )
Arguments: none
Result: handle = handle to font engine

OpenFont
Description: opens a system font for use
Library: graphics.library
Offset: -$48

Syntax: font = OpenFont(textAttr)
C: struct TextFont *OpenFont(struct TextAttr *)
ML: d0 = OpenFont(a0)
Arguments: textAttr = TextAttr or TTextAttr structure describing the desired font
Result: font = opened font; zero if unsuccessful

OpenFromLock (Revision 2.0)

Description: opens a file you have a lock on
Library: dos.library
Offset: -$17A
Syntax: fh = OpenFromLock(lock)
C: BPTR OpenFromLock(BPTR)
ML: d0 = OpenFromLock(d1)
Arguments: lock = lock on file to be opened
Result: fh = filehandle of open file; zero if unsuccessful

OpenIFF (Revision 2.0)

Description: prepares an IFFHandle to read or write a new IFF stream
Library: iffparse.library
Offset: -$24
Syntax: error = OpenIFF(iff, rwMode)
C: LONG OpenIFF(struct IFFHandle *, LONG)
ML: d0 = OpenIFF(a0, d0)
Arguments: iff = pointer to IFFHandle structure
rwMode = mode for I/O—IFFF_READ ($00000000), IFFF_WRITE ($00000001)
Result: error = zero if successful; nonzero error-code if unsuccessful

OpenLibrary

Description: gains access to a library
Library: exec.library
Offset: -$198
Syntax: library = OpenLibrary(libName, version)
C: struct Library *OpenLibrary(STRPTR, ULONG)
ML: d0 = OpenLibrary(a1, d0)
Arguments: libName = the name of the library to open
Result: library = the library base address; zero if unsuccessful

OpenLocale (Revision 3.0)

Description: opens a locale
Library: locale.library
Offset: -$9C
Syntax: locale = OpenLocale(name)
C: struct Locale *OpenLocale(STRPTR)
ML: d0 = OpenLocale(a0)

Arguments: name = the locale to open; 0 indicates the default locale
Result: locale = an initialized Locale structure; zero if unsuccessful

OpenMonitor (Revision 2.0)
Description: opens a MonitorSpec
Library: graphics.library
Offset: -$2CA
Syntax: mspc = OpenMonitor(monitorName , displayID)
C: struct MonitorSpec *OpenMonitor(char *, ULONG)
ML: d0 = OpenMonitor(a1, d0)
Arguments: monitorName = monitor name
displayID = optional monitor/mode identifier
Result: mspc = opened MonitorSpec structure; zero if unsuccessful

OpenResource
Description: gains access to a resource
Library: exec.library
Offset: -$1F2
Syntax: resource = OpenResource(resName)
C: APTR OpenResource(STRPTR)
ML: d0 = OpenResource(a1)
Arguments: resName = the name of the resource requested
Result: resource = the resource base pointer; zero if unsuccessful

OpenScreen
Description: opens an Intuition screen
Library: intuition.library
Offset: -$C6
Syntax: screen = OpenScreen(newScreen)
C: struct Screen *OpenScreen(struct ExtNewScreen *)
ML: d0 = OpenScreen(a0)
Arguments: newScreen = NewScreen structure that describes the screen to open
Result: screen = a new Screen structure; zero if unsuccessful

OpenScreenTagList (Revision 2.0)
Description: opens an Intuition screen with a TagItem extension array
Library: intuition.library
Offset: -$264
Syntax: screen = OpenScreenTagList(newScreen, tags)
C: struct Screen *OpenScreenTags(struct NewScreen *, struct TagItem *)
ML: d0 = OpenScreenTagList(a0, a1)
Arguments: newScreen = NewScreen structure; optional
tags = tag list of attributes—see <intuition/screens.h> for description
Result: screen = a new Screen structure; zero if unsuccessful

OpenWindow

Description: opens an Intuition window
Library: intuition.library
Offset: -$CC
Syntax: window = OpenWindow(newWindow)
C: struct Window *OpenWindow(struct NewWindow *)
ML: d0 = OpenWindow(a0)
Arguments: newWindow = NewWindow structure that describes window to open
Result: window = a new Window structure; zero if unsuccessful

OpenWindowTagList (Revision 2.0)

Description: opens an Intuition window with a TagItem extension array
Library: intuition.library
Offset: -$25E
Syntax: window = OpenWindowTagList(newWindow, tags)
C: struct Window *OpenWindowTags(struct NewWindow *, struct TagItem *)
ML: d0 = OpenWindowTagList(a0, a1)
Arguments: newWindow = NewWindow structure; optional
tags = tag list of attributes:
see <intuition/intuition.h> for description
Result: window = a new Window structure; zero if unsuccessful

OpenWorkBench

Description: opens the Workbench screen
Library: intuition.library
Offset: -$D2
Syntax: screen = OpenWorkBench( )
C: ULONG OpenWorkBench(VOID)
ML: d0 = OpenWorkBench( )
Arguments: none
Result: screen = the Workbench screen structure; zero if unsuccessful

OrRectRegion

Description: ORs a rectangle with a region, leaving the result in the region
Library: graphics.library
Offset: -$1FE
Syntax: success = OrRectRegion(region, rectangle)
C: BOOL OrRectRegion(struct Region *, struct Rectangle *)
ML: d0 = OrRectRegion(a0, a1)
Arguments: region = Region structure
rectangle = Rectangle structure
Result: success = zero if unsuccessful

OrRegionRegion

Description: ORs two regions together, leaving the result in the second region
Library: graphics.library
Offset: -$264
Syntax: success = OrRegionRegion(region1, region2)
C: BOOL OrRegionRegion(struct Region *, struct Region *)
ML: d0 = OrRegionRegion(a0, a1)
Arguments: region1 = Region structure
region2 = Region structure
Result: success = zero if unsuccessful

Output

Description: finds the program's initial output filehandle
Library: dos.library
Offset: -$3C
Syntax: fh = Output( )
C: BPTR Output(VOID)
ML: d0 = Output( )
Arguments: none
Result: fh = program's output filehandle

OwnBlitter

Description: reserves the blitter for private usage
Library: graphics.library
Offset: -$1C8
Syntax: OwnBlitter( )
C: VOID OwnBlitter(VOID)
ML: OwnBlitter( )
Arguments: none
Result: none

PackBoolTags (Revision 2.0)

Description: builds a flag word from a tag list
Library: utility.library
Offset: -$2A
Syntax: flags = PackBoolTags(initialFlags, tags, boolMap)
C: ULONG PackBoolTags(ULONG, struct TagItem *, struct TagItem *)
ML: d0 = PackBoolTags(d0, a0, a1)
Arguments: initialFlags = bit array to be modified
tags = the tag list to be scanned
boolMap = a tag list defining the tags to process; specifies the bits to be set in initialFlags
Result: flags = the accumulated longword of flags

PackStructureTags (Revision 3.0)

Description: packs a structure using values from a tag list
Library: utility.library
Offset: -$D2
Syntax: num = PackStructureTags(pack, packtable, tags)
C: ULONG PackStructureTags(APTR, ULONG *, struct TagItem *)
ML: d0 = PackStructureTags(a0, a1, a2)
Arguments: pack = the data area to initialize
packtable = the packing information table—see <utility/pack.h> for description
tags = the tag list to pack into the structure
Result: num = number of tag items packed

ParentChunk (Revision 2.0)

Description: gets the nesting context node for the given chunk
Library: iffparse.library
Offset: -$B4
Syntax: parent = ParentChunk(contextNode)
C: struct ContextNode *ParentChunk(struct ContextNode *)
ML: d0 = ParentChunk(a0)
Arguments: contextNode = a context node
Result: parent = the enclosing context node; zero if none exists

ParentDir

Description: obtains the parent directory of a file or directory
Library: dos.library
Offset: -$D2
Syntax: newlock = ParentDir(lock)
C: BPTR ParentDir(BPTR)
ML: d0 = ParentDir(d1)
Arguments: lock = lock on a file or directory
Result: newlock = lock on parent directory

ParentOfFH (Revision 2.0)

Description: obtains the parent directory of a file or directory
Library: dos.library
Offset: -$180
Syntax: lock = ParentOfFH(fh)
C: BPTR ParentOfFH(BPTR)
ML: d0 = ParentOfFH(d1)
Arguments: fh = filehandle of file or directory
Result: lock = lock on parent directory

ParseDate (Revision 3.0)

Description: interprets a string according to the date format of a locale
Library: locale.library

Offset: -$A2
Syntax: match = ParseDate(locale, date, template, getCharFunc)
C: BOOL ParseDate(struct Locale *, struct DateStamp *, STRPTR, struct Hook *)
ML: d0 = ParseDate(a0, a1, a2, a3)
Arguments: locale = the locale to use for the formatting
date = buffer to store the converted date; 0 to check date only
template = a template describing input format
getCharFunc = callback function called to obtain characters
Result: match = nonzero if the input matched the template

ParseIFF (Revision 2.0)

Description: parses an IFF file from an IFFHandle structure stream
Library: iffparse.library
Offset: -$2A
Syntax: error = ParseIFF(iff, control)
C: LONG ParseIFF(struct IFFHandle *, LONG)
ML: d0 = ParseIFF(a0, d0)
Arguments: iff = pointer to an IFFHandle structure
control = control code—IFFPARSE_SCAN ($00000000), IFFPARSE_STEP ($00000001), IFFPARSE_RAWSTEP ($00000002)
Result: error = zero if successful; nonzero error-code if unsuccessful

ParseIX (Revision 2.0)

Description: initializes an input expression
Library: commodities.library
Offset: -$84
Syntax: result = ParseIX(description, ix)
C: LONG ParseIX(STRPTR, IX *)
ML: d0 = ParseIX(a0, a1)
Arguments: description = string to parse
ix = input expression to hold result
Result: result = zero if successful; -1 if extra tokens, -2 if description was zero

ParsePattern (Revision 2.0)

Description: creates a tokenized pattern string for MatchPattern( )
Library: dos.library
Offset: -$348
Syntax: isWild = ParsePattern(source, dest, destLength)
C: LONG ParsePattern(STRPTR, STRPTR, LONG)
ML: d0 = ParsePattern(d1, d2, d3)

Arguments: source = unparsed string to find; may include wildcards
dest = buffer to contain tokenized pattern string
destLength = size of buffer; should be at least twice as large as the source string, plus four
Result: isWild = 1 if there were wildcards in the pattern;
0 if there were no wildcards in the pattern;
-1 if unsuccessful

ParsePatternNoCase (Revision 2.0)

Description: creates a tokenized pattern string for MatchPatternNoCase( )
Library: dos.library
Offset: -$3C0
Syntax: isWild = ParsePatternNoCase(source, dest, destLength)
C: LONG ParsePatternNoCase(STRPTR, STRPTR, LONG)
ML: d0 = ParsePatternNoCase(d1, d2, d3)
Arguments: source = unparsed string to find; may include wildcards
dest = buffer to contain tokenized pattern string
destLength = size of buffer; should be at least twice as large as the source string, plus four
Result: isWild = 1 if there were wildcards in the pattern;
0 if there were no wildcards in the pattern;
-1 if unsuccessful

PathPart (Revision 2.0)

Description: returns a pointer to the end of the next-to-last part of a pathname
Library: dos.library
Offset: -$36C
Syntax: fileptr = PathPart(path)
C: STRPTR PathPart(STRPTR)
ML: d0 = PathPart(d1)
Arguments: path = pathname
Result: fileptr = pointer to the end of the next-to-last part of the pathname

Permit

Description: enables task rescheduling
Library: exec.library
Offset: -$8A
Syntax: Permit( )
C: VOID Permit(VOID)
ML: Permit( )
Arguments: none
Result: none

PointInImage (Revision 2.0)

Description: tests whether an image contains a specific point
Library: intuition.library
Offset: -$270
Syntax: doesContain = PointInImage(point, image)
C: BOOL PointInImage(struct Point, struct Image *)
ML: d0 = PointInImage(d0, a0)
Arguments: point = the point's coordinates
image = image to check
Result: doesContain = nonzero if point is found in image definition or
image = 0

PolyDraw

Description: draws a polygon from an array of points
Library: graphics.library
Offset: -$150
Syntax: PolyDraw(rastPort, count , array)
C: VOID PolyDraw(struct RastPort *, WORD, WORD *)
ML: PolyDraw(a1, d0, a0)
Arguments: rastPort = RastPort structure
count = number of points in the array
array = array of points
Result: none

PopChunk (Revision 2.0)

Description: pops the top context node off the context stack
Library: iffparse.library
Offset: -$5A
Syntax: error = PopChunk(iff)
C: LONG PopChunk(struct IFFHandle *)
ML: d0 = PopChunk(a0)
Arguments: iff = an IFFHandle structure
Result: error = zero if successful; nonzero error-code if unsuccessful

PrintDTObjectA (Revision 3.0)

Description: calls a DataType object's DTM_PRINT method as a separate process
Library: datatypes.library
Offset: -$72
Syntax: success = PrintDTObjectA(object, window, requester, msg)
C: ULONG PrintDTObjectA(Object *, struct Window *, struct Requester *, struct dtPrint *)
ML: d0 = PrintDTObjectA(a0, a1, a2, a3)

Arguments: object = Object structure returned by NewDTObjectA( )
window = window to which the object is attached
requester = requester to which the object is attached
msg = Initialized dtPrint structure
Result: success = zero if unsuccessful

PrintFault (Revision 2.0)
Description: returns the text associated with a DOS error code
Library: dos.library
Offset: -$1DA
Syntax: success = PrintFault(code, header)
C: BOOL PrintFault(LONG, STRPTR)
ML: d0 = PrintFault(d1, d2)
Arguments: code = error code
header = text to output before error text
Result: success = zero if unsuccessful

PrintIText
Description: prints text found in an IntuiText structure
Library: intuition.library
Offset: -$D8
Syntax: PrintIText(rastPort, iText, leftOffset, topOffset)
C: VOID PrintIText(struct RastPort *, struct IntuiText *, WORD, WORD)
ML: PrintIText(a0, a1, d0, d1)
Arguments: rastPort = destination RastPort
iText = IntuiText structure containing text
leftOffset = horizontal offset from left edge of RastPort
topOffset = vertical offset from top edge of RastPort
Result: none

Procure (Revision 3.0)
Description: makes a bid for a semaphore
Library: exec.library
Offset: -$21C
Syntax: Procure(semaphore, bidMessage)
C: VOID Procure(struct SignalSemaphore *, struct SemaphoreMessage *)
ML: Procure(a0, a1)
Arguments: semaphore = the SignalSemaphore desired
bidMessage = the SemaphoreMessage to be sent when the
semaphore is obtained
Result: none

PropChunk (Revision 2.0)

Description: stores a property chunk
Library: iffparse.library
Offset: -$72
Syntax: error = PropChunk(iff, type, id)
C: LONG PropChunk(struct IFFHandle *, LONG, LONG)
ML: d0 = PropChunk(a0, d0, d1)
Arguments: iff = an IFFHandle structure
type = type code for the new chunk
id = identifier for the new chunk
Result: error = zero if successful; nonzero error-code if unsuccessful

PropChunks (Revision 2.0)

Description: declares multiple property chunks
Library: iffparse.library
Offset: -$78
Syntax: error = PropChunks(iff, propArray, numPairs)
C: LONG PropChunks(struct IFFHandle *, LONG *, LONG)
ML: d0 = PropChunks(a0, a1, d0)
Arguments: iff = an IFFHandle structure
propArray = array of longword chunk types and identifiers
numPairs = number of pairs in propArray
Result: error = zero if successful; nonzero error-code if unsuccessful

PubScreenStatus (Revision 2.0)

Description: changes a public screen's status flags
Library: intuition.library
Offset: -$228
Syntax: resultFlags = PubScreenStatus(screen, statusFlags)
C: UWORD PubScreenStatus(struct Screen *, UWORD)
ML: d0 = PubScreenStatus(a0, d0)
Arguments: screen = public screen to change
statusFlags = new flags—PSNF_PRIVATE ($00000001)
Result: resultFlags = zero if unsuccessful

PushChunk (Revision 2.0)

Description: pushes a new context node on the context stack
Library: iffparse.library
Offset: -$54
Syntax: error = PushChunk(iff, type, id, size);
C: LONG PushChunk(struct IFFHandle *, LONG, LONG, LONG)
ML: d0 = PushChunk(a0, d0, d1, d2)

Arguments: iff = an IFFHandle structure
type = chunk type specifier
id = chunk id specifier
size = size of the chunk to create or IFFSIZE_UNKNOWN ($FFFFFFFF)
Result: error = zero if successful; nonzero error-code if unsuccessful

PutDefDiskObject (Revision 2.0)
Description: writes a disk object as the default for its type
Library: icon.library
Offset: -$7E
Syntax: success = PutDefDiskObject(diskobj)
C: BOOL PutDefDiskObject(struct DiskObject *)
ML: d0 = PutDefDiskObject(a0)
Arguments: diskobj = DiskObject structure to write
Result: success = zero if unsuccessful

PutDiskObject
Description: writes a DiskObject to disk
Library: icon.library
Offset: -$54
Syntax: success = PutDiskObject(name, diskobj)
C: BOOL PutDiskObject(char *, struct DiskObject *)
ML: d0 = PutDiskObject(a0, a1)
Arguments: name = name of the object to write
diskobj = DiskObject structure to write
Result: success = zero if unsuccessful

PutMsg
Description: puts a message to a message port
Library: exec.library
Offset: -$16E
Syntax: PutMsg(port, message)
C: VOID PutMsg(struct MsgPort *, struct Message *)
ML: PutMsg(a0, a1)
Arguments: port = the message port to receive the message
message = the message to put
Result: none

PutStr (Revision 2.0)
Description: writes a string to the the default output device
Library: dos.library
Offset: -$3B4
Syntax: error = PutStr(str)
C: LONG PutStr(STRPTR)

ML: d0 = PutStr(d1)
Arguments: str = string to write
Result: error = nonzero if unsuccessful

QBlit

Description: queues a blitter request
Library: graphics.library
Offset: -$114
Syntax: QBlit(bp)
C: VOID QBlit(struct bltnode *)
ML: QBlit(a1)
Arguments: bp = initialized bltnode structure
Result: none

QBSBlit

Description: queues a blitter request that is linked to the video beam's position
Library: graphics.library
Offset: -$126
Syntax: QBSBlit(bsp)
C: VOID QBSBlit(struct bltnode *)
ML: QBSBlit(a1)
Arguments: bp = initialized bltnode structure
Result: none

QueryOverscan (Revision 2.0)

Description: checks how large a system-legal overscan screen may be
Library: intuition.library
Offset: -$1DA
Syntax: success = QueryOverscan(displayID, rect, oScanType)
C: LONG QueryOverscan(ULONG, struct Rectangle *, WORD)
ML: d0 = QueryOverscan(a0, a1, d0)
Arguments: displayID = desired display mode—see <graphics/displaymodes.h>
rect = Rectangle structure to receive data
oScanType = type of overscan mode—OSCAN_TEXT ($0001), OSCAN_STANDARD ($0002), OSCAN_MAX ($0003), OSCAN_VIDEO ($0004)
Result: success = zero if unsuccessful

RawDoFmt

Description: formats data into a character stream
Library: exec.library
Offset: -$20A
Syntax: nextData = RawDoFmt(formatString, dataStream, putChProc, putChData);
C: APTR RawDoFmt(STRPTR, APTR, VOID (*)( ), APTR)
ML: d0 = RawDoFmt(a0, a1, a2, a3)

Arguments: formatString = a template string similar to that used by C's printf( )
dataStream = the data to format
putChProc = function called to output each character
putChData = user data passed to the putChProc along with each character
Result: nextData = a pointer to the end of dataStream; valid under Revision 2.0 only

Read

Description: reads bytes of data from a file
Library: dos.library
Offset: -$2A
Syntax: actualLength = Read(fh, buffer, length)
C: LONG Read(BPTR, VOID *, LONG)
ML: d0 = Read(d1, d2, d3)
Arguments: fh = filehandle of file from which to read
buffer = input buffer to receive data
length = number of bytes to read; may not exceed buffer size
Result: actualLength = actual number of bytes read

ReadArgs (Revision 2.0)

Description: parses command line input
Library: dos.library
Offset: -$31E
Syntax: result = ReadArgs(template, array, rdargs)
C: struct RDArgs *ReadArgs(STRPTR, LONG *, struct RDArgs *)
ML: d0 = ReadArgs(d1, d2, d3)
Arguments: template = format string
array = array of longwords to hold results, one per template entry
rdargs = optional rdargs structure (as returned by AllocDosObject( )) to hold options
Result: result = an initialized RDArgs structure; zero if unsuccessful

ReadChunkBytes (Revision 2.0)

Description: reads bytes from the current chunk
Library: iffparse.library
Offset: -$3C
Syntax: actual = ReadChunkBytes(iff, buf, numBytes)
C: LONG ReadChunkBytes(struct IFFHandle *, APTR, LONG)
ML: d0 = ReadChunkBytes(a0, a1, d0)
Arguments: iff = an IFFHandle structure
buf = buffer to receive data
numBytes = number of bytes to read
Result: actual = number of bytes actually read, if successful;
negative error value if unsuccessful

ReadChunkRecords (Revision 2.0)

Description: reads record elements from the current chunk
Library: iffparse.library
Offset: -$48
Syntax: actual = ReadChunkRecords(iff, buf, bytesPerRecord, numRecords)
C: LONG ReadChunkRecords(struct IFFHandle *, APTR, LONG, LONG)
ML: d0 = ReadChunkRecords(a0, a1, d0, d1)
Arguments: iff = an IFFHandle structure
buf = buffer to receive data
bytesPerRecord = size of data records to read
numRecords = number of data records to read
Result: actual = number of whole records read if successful; negative if an error occurs

ReadExpansionByte

Description: reads a byte from expansion memory
Library: expansion.library
Offset: -$60
Syntax: byte = ReadExpansionByte(board, offset)
C: UBYTE ReadExpansionByte(APTR, ULONG)
ML: d0 = ReadExpansionByte(a0, d0)
Arguments: board = base pointer to a new style expansion board
offset = logical byte offset from the board base
Result: byte = byte of data from the expansion board.

ReadExpansionRom

Description: reads an expansion board's configuration ROM space
Library: expansion.library
Offset: -$66
Syntax: ReadExpansionRom(board, configDev)
C: VOID ReadExpansionRom(APTR, struct ConfigDev *)
ML: ReadExpansionRom(a0, a1)
Arguments: board = base pointer to a new style expansion board
configDev = ConfigDev structure to receive data
Result: none

ReadItem (Revision 2.0)

Description: reads a single argument from command line
Library: dos.library
Offset: -$32A
Syntax: value = ReadItem(buffer, maxchars, input)
C: LONG ReadItem(STRPTR, LONG, struct CSource *)
ML: d0 = ReadItem(d1, d2, d3)

Arguments: buffer = buffer to receive command line argument
maxchars = size of buffer
input = CSource input or NULL (uses FGetC(Input( )))
Result: value = 0 if end of command line; 1 if item is not in quotes; 2 if item is in quotes; -1 if unsuccessful

ReadLink (Revision 2.0)

Description: reads the path for a soft filesystem link
Library: dos.library
Offset: -$1B6
Syntax: success = ReadLink(port, lock, path, buffer, size)
C: BOOL ReadLink(struct MsgPort *, BPTR, STRPTR, STRPTR, ULONG)
ML: d0 = ReadLink(d1, d2, d3, d4, d5)
Arguments: port = msgport of the filesystem
lock = lock to which this path is relative
path = path that caused the ERROR_IS_SOFT_LINK
buffer = buffer to receive new path
size = size of buffer
Result: Success = zero if unsuccessful

ReadPixel

Description: returns the color of a specific pixel
Library: graphics.library
Offset: -$13E
Syntax: penno = ReadPixel(rastPort, x, y)
C: LONG ReadPixel(struct RastPort *, WORD, WORD)
ML: d0 = ReadPixel(a1, d0, d1)
Arguments: rastPort = RastPort structure
x, y = coordinate of pixel in question
Result: penno = pixel color; -1 if unsuccessful

ReadPixelArray8 (Revision 2.0)

Description: reads the pixel colors from a rectangle
Library: graphics.library
Offset: -$30C
Syntax: count = ReadPixelArray8(rastPort, x1, y1, x2, y2, array, temprastPort)
C: LONG ReadPixelArray8(struct RastPort *, UWORD, UWORD, UWORD, UWORD, UBYTE *, struct RastPort *)
ML: d0 = ReadPixelArray8(a0, d0, d1, d2, d3, a2, a1)
Arguments: rastPort = RastPort structure
x1, y1 = top-left point in the RastPort
x2, y2 = bottom-right point in the RastPort
array = array to hold pixel color values
temprastPort = temporary RastPort structure
Result: count = number of pixels read

ReadPixelLine8 (Revision 2.0)

Description: reads the pixel colors from a raster line
Library: graphics.library
Offset: -$300
Syntax: count = ReadPixelLine8(rastPort, x1, y1, width, array, tempRastPort)
C: LONG ReadPixelLine8(struct RastPort *, UWORD, UWORD, UWORD, UBYTE *, struct RastPort *)
ML: d0 = ReadPixelLine8(a0, d0, d1, d2, a2, a1)
Arguments: rastPort = RastPort structure
x, y = starting point of line
width = number of horizontal pixels to read
array = array to hold pixel color values
tempRastPort = temporary RastPort structure
Result: count = number of pixels read

RectFill

Description: draws a filled rectangle
Library: graphics.library
Offset: -$132
Syntax: RectFill(rastPort, x1, y1, x2, y2)
C: VOID RectFill(struct RastPort *, WORD, WORD, WORD, WORD)
ML: RectFill(a1, d0, d1, d2, d3)
Arguments: rastPort = RastPort structure
x1, y1 = top-left point of rectangle
x2, y2 = bottom-right point of rectangle
Result: none

RefreshDTObjectA (Revision 3.0)

Description: refreshes a DataType object
Library: datatypes.library
Offset: -$4E
Syntax: RefreshDTObjectA(object, window, requester, tags)
C: VOID RefreshDTObjectA(Object *, struct Window *, struct Requester *, struct TagItem *)
ML: d0 = RefreshDTObjectA(a0, a1, a2, a3)
Arguments: object = Object structure returned by NewDTObjectA( )
window = window to which the object is attached
requester = requester to which the object is attached
tags = tag list specifying addition attributes—none are currently defined
Result: none

RefreshGadgets

Description: redraws a gadget and all subsequent gadgets in a gadget list
Library: intuition.library
Offset: -$DE

Syntax: RefreshGadgets(gadgets, window, requester)
C: VOID RefreshGadgets(struct Gadget *, struct Window *, struct Requester *)
ML: RefreshGadgets(a0, a1, a2)
Arguments: gadgets = first gadget in list of gadgets to refresh
window = window containing gadgets
requester = requester containing gadget; may be 0
Result: none

RefreshGList

Description: redraws a gadget and a specified number of subsequent gadgets in a gadget list
Library: intuition.library
Offset: -$1B0
Syntax: RefreshGList(gadgets, window, requester, numGad)
C: VOID RefreshGList(struct Gadget *, struct Window *, struct Requester *, WORD)
ML: RefreshGList(a0, a1, a2, d0)
Arguments: gadgets = first gadget in list of gadgets to refresh
window = window containing gadgets
requester = requester containing gadget; may be 0
numGad = number of gadgets to refresh; -1 to refresh to end of gadget list
Result: none

RefreshTagItemClones (Revision 2.0)

Description: rejuvenates a cloned tag list from the original
Library: utility.library
Offset: -$54
Syntax: RefreshTagItemClones(clone, original)
C: VOID RefreshTagItemClones(struct TagItem *, struct TagItem *)
ML: RefreshTagItemClones(a0, a1)
Arguments: clone = tag list returned by CloneTagItems( )
original = tag list originally passed to CloneTagItems( )
Result: none

RefreshWindowFrame

Description: redraws a window's border, title bar, and gadgets
Library: intuition.library
Offset: -$1C8
Syntax: RefreshWindowFrame(window)
C: VOID RefreshWindowFrame(struct Window *)
ML: RefreshWindowFrame(a0)
Arguments: window = window to refresh
Result: none

Relabel (Revision 2.0)

Description: changes the name of a volume
Library: dos.library
Offset: -$2D0
Syntax: success = Relabel(volumename, name)
C: BOOL Relabel(STRPTR, STRPTR)
ML: d0 = Relabel(d1, d2)
Arguments: volumename = current name of device to rename (with :)
newname = new volume name (without :)
Result: success = zero if unsuccessful

ReleaseConfigBinding

Description: allows others to bind to drivers
Library: expansion.library
Offset: -$7E
Syntax: ReleaseConfigBinding( )
C: VOID ReleaseConfigBinding(VOID)
ML: ReleaseConfigBinding( )
Arguments: none
Result: none

ReleaseDataType (Revision 3.0)

Description: releases a DataType structure returned by ObtainDataTypeA( )
Library: datatypes.library
Offset: -$2A
Syntax: ReleaseDataType(dtn)
C: VOID ReleaseDataType(struct DataType *)
ML: ReleaseDataType(a0)
Arguments: dtn = DataType structure returned by ObtainDataTypeA( )
Result: none

ReleaseGIRPort (Revsion 2.0)

Description: releases a custom gadget from a RastPort
Library: intuition.library
Offset: -$234
Syntax: ReleaseGIRPort(rastPort)
C: VOID ReleaseGIRPort(struct RastPort *)
ML: ReleaseGIRPort(a0)
Arguments: rastPort = RastPort with related custom gadget
Result: none

ReleaseInfoA (Revision 3.0)

Description: releases data obtained with ObtainInfoA( )
Library: bullet.library
Offset: -$36

Syntax: success = ReleaseInfoA(handle, tags)
C: ULONG ReleaseInfo(struct GlyphEngine *, struct TagItem *)
ML: d0 = ReleaseInfoA(a0, a1)
Arguments: handle = handle returned by OpenEngine( )
tags = tag list specifying data to release—see <diskfont diskfonttags.h> for description
Result: none

ReleaseNamedObject (Revision 3.0)
Description: releases a name object and decrements its usage count
Library: utility.library
Offset: -$102
Syntax: ReleaseNamedObject(object)
C: VOID ReleaseNamedObject(struct NamedObject *)
ML: ReleaseNamedObject(a0)
Arguments: object = the object to release
Result: none

ReleasePen (Revision 3.0)
Description: releases a palette entry obtained via the ObtainPen( ) function
Library: graphics.library
Offset: -$3B4
Syntax: ReleasePen(colorMap, n)
C: VOID ReleasePen(struct ColorMap *, ULONG)
ML: ReleasePen(a0, d0)
Arguments: colorMap = colormap
pen = palette entry to release
Result: none

ReleaseSemaphore
Description: makes signal semaphore available
Library: exec.library
Offset: -$23A
Syntax: ReleaseSemaphore(signalSemaphore)
C: VOID ReleaseSemaphore(struct SignalSemaphore *)
ML: ReleaseSemaphore(a0)
Arguments: signalSemaphore = an initialized signal semaphore structure
Result: none

ReleaseSemaphoreList
Description: makes a list of semaphores available
Library: exec.library
Offset: -$24C
Syntax: ReleaseSemaphoreList(list)
C: VOID ReleaseSemaphoreList(struct List *)

ML: ReleaseSemaphoreList(a0)
Arguments: list = a list of signal semaphores
Result: none

RemakeDisplay

Description: redraws all open Intuition screens
Library: intuition.library
Offset: -$180
Syntax: RemakeDisplay( )
C: VOID RemakeDisplay(VOID)
ML: RemakeDisplay( )
Arguments: none
Result: none

RemAssignList (Revision 2.0)

Description: removes an entry from a multi-directory assign
Library: dos.library
Offset: -$27C
Syntax: success = RemAssignList(name, lock)
C: BOOL RemAssignList(STRPTR, BPTR)
ML: d0 = RemAssignList(d1, d2)
Arguments: name = name of device from which to remove lock (without :)
lock = lock associated with the object to remove
Result: success = zero if unsuccessful

RemConfigDev

Description: removes a ConfigDev structure from the system
Library: expansion.library
Offset: -$6C
Syntax: RemConfigDev(configDev)
C: VOID RemConfigDev(struct ConfigDev *)
ML: RemConfigDev(a0)
Arguments: configDev = a valid ConfigDev structure
Result: none

RemDevice

Description: removes a device from the system
Library: exec.library
Offset: -$1B6
Syntax: RemDevice(device)
C: VOID RemDevice(struct Device *)
ML: RemDevice(a1)
Arguments: device = the device to remove
Result: none

RemDosEntry (Revision 2.0)
Description: removes a DosList entry from its list
Library: dos.library
Offset: -$2A0
Syntax: success = RemDosEntry(dlist)
C: BOOL RemDosEntry(struct DosList *)
ML: d0 = RemDosEntry(d1)
Arguments: dlist = device list entry to remove
Result: success = zero if unsuccessful

RemFont
Description: removes a font from the system list
Library: graphics.library
Offset: -$1E6
Syntax: RemFont(textFont)
C: VOID RemFont(struct TextFont *)
ML: RemFont(a1)
Arguments: textFont = TextFont structure to remove
Result: none

RemHead
Description: removes the head node from a list
Library: exec.library
Offset: -$102
Syntax: node = RemHead(list)
C: struct Node *RemHead(struct List *)
ML: d0 = RemHead(a0)
Arguments: list = the target list header
Result: node = the node removed; zero if the list is empty

RemIBob
Description: removes a Bob from the gel list
Library: graphics.library
Offset: -$84
Syntax: RemIBob(bob, rastPort, viewPort)
C: VOID RemIBob(struct Bob *, struct RastPort *, struct ViewPort *)
ML: RemIBob(a0, a1, a2)
Arguments: bob = Bob to be removed
rastPort = RastPort structure
viewPort = ViewPort structure
Result: none

RemIntServer
Description: removes an interrupt server from a server chain
Library: exec.library

Offset: -$AE
Syntax: RemIntServer(intNum, interrupt)
C: VOID RemIntServer(ULONG, struct Interrupt *)
ML: RemIntServer(d0, a1)
Arguments: intNum = the Paula interrupt bit; 0-14
interrupt = the interrupt server to remove
Result: none

RemLibrary

Description: removes a library from the system
Library: exec.library
Offset: -$192
Syntax: RemLibrary(library)
C: VOID RemLibrary(struct Library *)
ML: RemLibrary(a1)
Arguments: library = the library to remove
Result: none

RemMemHandler (Revision 3.0)

Description: removes a low memory handler from exec
Library: exec.library
Offset: -$30C
Syntax: RemMemHandler(memHandler)
C: VOID RemMemHandler(struct Interrupt *)
ML: RemMemHandler(a1)
Arguments: memHandler = a handler added with AddMemHandler( )
Result: none

RemNamedObject (Revision 3.0)

Description: requests the removal of a named object from a namespace
Library: utility.library
Offset: -$108
Syntax: RemNamedObject(object, message)
C: VOID RemNamedObject(struct NamedObject *, struct Message *)
ML: RemNamedObject(a0, a1)
Arguments: object = the object to remove
message = message to pass to ReplyMsg( )
Result: none

Remove

Description: removes a node from a list
Library: exec.library
Offset: -$FC
Syntax: Remove(node)
C: VOID Remove(struct Node *)

ML: Remove(a1)
Arguments: node = the node to remove
Result: none

RemoveAmigaGuideHostA (Revision 3.0)
Description: removes a host
Library: amigaguide.library
Offset: -$90
Syntax: use = RemoveAmigaGuideHostA(handle, tags)
C: LONG RemoveAmigaGuideHostA(AMIGAGUIDEHOST, struct TagItem *)
ML: d0 = RemoveAmigaGuideHostA(a0, a1)
Arguments: handle = value returned from AddAmigaGuideHost( )
tags = tag list of attributes—none are currently defined
Result: use = number of outstanding clients

RemoveAppIcon (Revision 2.0)
Description: removes an AppIcon
Library: workbench.library
Offset: -$42
Syntax: success = RemoveAppIcon(AppIcon)
C: BOOL RemoveAppIcon(struct AppIcon *)
ML: d0 = RemoveAppIcon(a0)
Arguments: AppIcon = AppIcon structure returned by AddAppIcon( )
Result: success = always returns 1

RemoveAppMenuItem (Revision 2.0)
Description: removes an AppMenuItem
Library: workbench.library
Offset: -$4E
Syntax: success = RemoveAppMenuItem(AppMenuItem)
C: BOOL RemoveAppMenuItem(struct AppMenuItem *)
ML: d0 = RemoveAppMenuItem(a0)
Arguments: AppMenuItem = AppMenuItem structure returned by
AddAppMenuItem( )
Result: success = always returns 1

RemoveAppWindow (Revision 2.0)
Description: removes an AppWindow
Library: workbench.library
Offset: -$36
Syntax: success = RemoveAppWindow(AppWindow)
C: BOOL RemoveAppWindow(struct AppWindow *)
ML: d0 = RemoveAppWindow(a0)

Arguments: AppWindow = AppWindow structure returned by AddAppWindow( )
Result: success = always returns 1

RemoveClass (Revision 2.0)
Description: makes a public boopsi class unavailable
Library: intuition.library
Offset: -$2C4
Syntax: RemoveClass(classPtr)
C: VOID RemoveClass(struct IClass *)
ML: RemoveClass(a0)
Arguments: classPtr = public class created by MakeClass( ) function
Result: none

RemoveCxObj (Revision 2.0)
Description: removes a commodity object from a list
Library: commodities.library
Offset: -$66
Syntax: RemoveCxObj(cxobj)
C: VOID RemoveCxObj(CxObj *)
ML: RemoveCxObj(a0)
Arguments: cxobj = commodity object to remove
Result: none

RemoveDTObject (Revision 3.0)
Description: removes a DataType object from a window
Library: datatypes.library
Offset: -$60
Syntax: position = RemoveDTObject(window, object)
C: LONG RemoveDTObject(struct Window *, Object *)
ML: d0 = RemoveDTObject(a0, a1)
Arguments: window = window to which the object is attached
object = Object structure returned by NewDTObjectA( )
Result: position = position of the removed object in the window's gadget list; -1 if the object wasn't found

RemoveGadget
Description: removes a gadget from a window
Library: intuition.library
Offset: -$E4
Syntax: position = RemoveGadget(window, gadget)
C: UWORD RemoveGadget(struct Window *, struct Gadget *)
ML: d0 = RemoveGadget(a0, a1)

Arguments: window = window with unwanted gadget
gadget = unwanted gadget
Result: position = position of removed gadget; -1 if unsuccessful

RemoveGList
Description: removes a series of gadgets from a gadget list
Library: intuition.library
Offset: -$1BC
Syntax: position = RemoveGList(window, gadget, numgad)
C: UWORD RemoveGList(struct Window *, struct Gadget *, WORD)
ML: d0 = RemoveGList(a0, a1, d0)
Arguments: window = window with unwanted gadgets
gadget = first gadget in gadget list
numgad = number of gadgets to remove; -1 to remove all subsequent gadgets
Result: position = position of first removed gadget; -1 if unsuccessful

RemPort
Description: removes a message port from the system
Library: exec.library
Offset: -$168
Syntax: RemPort(port)
C: VOID RemPort(struct MsgPort *)
ML: RemPort(a1)
Arguments: port = the message port to remove
Result: none

RemResource
Description: removes a resource from the system
Library: exec.library
Offset: -$1EC
Syntax: RemResource(resource)
C: VOID RemResource(APTR)
ML: RemResource(a1)
Arguments: resource = the resource to remove
Result: none

RemSegment (Revision 2.0)
Description: removes a resident segment from the resident list
Library: dos.library
Offset: -$312
Syntax: success = RemSegment(segment)
C: BOOL RemSegment(struct Segment *)
ML: d0 = RemSegment(d1)
Arguments: segment = the segment to remove
Result: success = zero if unsuccessful

RemSemaphore

Description: removes a signal semaphore from the system
Library: exec.library
Offset: -$25E
Syntax: RemSemaphore(signalSemaphore)
C: VOID RemSemaphore(struct SignalSemaphore *)
ML: RemSemaphore(a1)
Arguments: signalSemaphore = an initialized signal semaphore structure
Result: none

RemTail

Description: removes the tail node from a list
Library: exec.library
Offset: -$108
Syntax: node = RemTail(list)
C: struct Node *RemTail(struct List *)
ML: d0 = RemTail(a0)
Arguments: list = the target list header
Result: node = the node removed; zero if the list is empty

RemTask

Description: removes a task from the system
Library: exec.library
Offset: -$120
Syntax: RemTask(task)
C: VOID RemTask(struct Task *)
ML: RemTask(a1)
Arguments: task = the task node to remove; 0 indicates self-removal
Result: none

RemVSprite

Description: removes a VSprite from the current gel list
Library: graphics.library
Offset: -$8A
Syntax: RemVSprite(vs)
C: VOID RemVSprite(struct VSprite *)
ML: RemVSprite(a0)
Arguments: vs = vsprite to remove
Result: none

Rename

Description: renames a directory or file
Library: dos.library
Offset: -$4E
Syntax: success = Rename(oldName, newName)

C: BOOL Rename(STRPTR, STRPTR)
ML: d0 = Rename(d1, d2)
Arguments: oldName = old name of file or directory
newName = new name for file or directory
Result: success = zero if unsuccessful

ReplyAmigaGuideMsg (Revision 3.0)
Description: replies to a message
Library: amigaguide.library
Offset: -$54
Syntax: ReplyAmigaGuideMsg(msg)
C: VOID ReplyAmigaGuideMsg(struct AmigaGuideMsg *msg)
ML: ReplyAmigaGuideMsg(a0)
Arguments: msg = message returned by GetAmigaGuideMsg( )
Result: none

ReplyMsg
Description: puts a message to its reply port
Library: exec.library
Offset: -$17A
Syntax: ReplyMsg(message)
C: VOID ReplyMsg(struct Message *)
ML: ReplyMsg(a1)
Arguments: message = the message to put
Result: none

ReplyPkt (Revision 2.0)
Description: replies a packet to the person who sent it to you
Library: dos.library
Offset: -$102
Syntax: ReplyPkt(packet, result1, result2)
C: VOID ReplyPkt(struct DosPacket *, LONG, LONG)
ML: ReplyPkt(d1, d2, d3)
Arguments: packet = packet to reply
result1 = first result
result2 = secondary result
Result: none

ReportMouse
Description: tells Intuition whether to report mouse movements related to a window
Library: intuition.library
Offset: -$EA
Syntax: ReportMouse(window, flag)
C: VOID ReportMouse(struct Window *, BOOL)
ML: ReportMouse(a0, d0)

Arguments: window = window to affect
flag = nonzero to activate mouse-movement messages; zero to stop such messages
Result: none

Request

Description: activates a requester
Library: intuition.library
Offset: -$F0
Syntax: success = Request(requester, window)
C: BOOL Request(struct Requester *, struct Window *)
ML: d0 = Request(a0, a1)
Arguments: requester = requester to display
window = window to contain requester
Result: success = zero if unsuccessful

RequestFile (Revision 2.0—Obsolete under Revision 3.0)

Description: prompts user for a filename
Library: asl.library
Offset: -$2A
Syntax: result = RequestFile(requester)
C: BOOL RequestFile(struct FileRequester *)
ML: d0 = RequestFile(a0)
Arguments: requester = FileRequester structure returned by AllocFileRequest( )
Result: result = zero if no filename selected, nonzero otherwise

ResetMenuStrip (Revision 2.0)

Description: reattaches a menu strip to a window
Library: intuition.library
Offset: -$2BE
Syntax: success = ResetMenuStrip(window, menu)
C: BOOL ResetMenuStrip(struct Window *, struct Menu *)
ML: d0 = ResetMenuStrip(a0, a1)
Arguments: window = window to receive menu strip
menu = menu strip to reattatch
Result: success = always nonzero

RethinkDisplay

Description: performs a complete overhaul on the Intuition display
Library: intuition.library
Offset: -$186
Syntax: RethinkDisplay( )
C: VOID RethinkDisplay(VOID)
ML: RethinkDisplay( )
Arguments: none
Result: none

RouteCxMsg (Revision 2.0)

Description: sets the next destination of a commodity message
Library: commodities.library
Offset: -$A2
Syntax: RouteCxMsg(cxm,cxobj)
C: VOID RouteCxMsg(struct CxMsg *, CxObj *)
ML: RouteCxMsg(a0, a1)
Arguments: cxm = commodity message to route
cxobj = destination commodity object
Result: none

RunCommand (Revision 2.0)

Description: runs a program using the current process
Library: dos.library
Offset: -$1F8
Syntax: retcode = RunCommand(seglist, stacksize, argptr, argsize)
C: LONG RunCommand(BPTR, ULONG, STRPTR, ULONG)
ML: d0 = RunCommand(d1, d2, d3, d4)
Arguments: seglist = Seglist of command to run
stacksize = number of bytes to allocate for stack space
argptr = command line string
argsize = number of characters in argptr string
Result: retcode = return code from executed command; -1 if unsuccessful

SameDevice (Revision 2.0)

Description: checks to see if two locks are on the same device
Library: dos.library
Offset: -$3D2
Syntax: same = SameDevice(lock1, lock2)
C: BOOL SameDevice(BPTR, BPTR)
ML: d0 = SameDevice(d1, d2)
Arguments: lock1, lock2 = locks
Result: same = nonzero if the locks are on the same device

SameLock (Revision 2.0)

Description: checks to see if two locks are on the same object
Library: dos.library
Offset: -$1A4
Syntax: value = SameLock(lock1, lock2)
C: LONG SameLock(BPTR, BPTR)
ML: d0 = SameLock(d1, d2)
Arguments: lock1, lock2 = locks
Result: value = 0 if the locks refer to the same object;
1 of locks are on the same volume;
-1 if locks refer to different objects

ScalerDiv (Revision 2.0)

Description: returns the scaling result that a call to BitMapScale( ) function would produce
Library: graphics.library
Offset: -$2AC
Syntax: result = ScalerDiv(factor, numerator, denominator)
C: UWORD ScalerDiv(UWORD, UWORD, UWORD)
ML: d0 = ScalerDiv(d0, d1, d2)
Arguments: factor = number in the range 0-16383
numerator, denominator = numbers in the range 1-16383
Result: result = factor*numerator/denominator

ScreenDepth (Revision 3.0)

Description: adjusts a screen's relative depth
Library: intuition.library
Offset: -$312
Syntax: ScreenDepth(screen, flags, reserved)
C: VOID ScreenDepth(struct Screen *, ULONG, APTR)
ML: ScreenDepth(a0, d0, a1)
Arguments: screen = screen to adjust
flags = desired depth setting:
SDEPTH_TOFRONT ($00000000), SDEPTH_TOBACK ($00000001),
SDEPTH_INFAMILY ($00000002)
Result: none

ScreenPosition (Revision 3.0)

Description: moves a screen
Library: intuition.library
Offset: -$318
Syntax: ScreenPosition(screen, flags, x1, y1, x2, y2)
C: VOID ScreenPosition(struct Screen *, ULONG, LONG, LONG, LONG, LONG)
ML: ScreenPosition(a0, d0, d1, d2, d3, d4)
Arguments: screen = screen to move
x1, y1 = new position of screen; may be absolute or relative
see flags, below
x2, y2 = describe the lower-right of the visible rectangle;
valid only with SPOS_MAKEVISIBLE
flags = defines how screen is moved
SPOS_RELATIVE ($00000000), SPOS_ABSOLUTE ($00000001),
SPOS_MAKEVISIBLE ($00000002), SPOS_FORCEDRAG ($00000004)
Result: none

ScreenToBack

Description: moves a screen behind all other screens
Library: intuition.library
Offset: -$F6

Syntax: ScreenToBack(screen)
C: VOID ScreenToBack(struct Screen *)
ML: ScreenToBack(a0)
Arguments: screen = screen to be pushed to back
Result: none

ScreenToFront
Description: moves a screen in front of all other screens
Library: intuition.library
Offset: -$FC
Syntax: ScreenToFront(screen)
C: VOID ScreenToFront(struct Screen *)
ML: ScreenToFront(a0)
Arguments: screen = screen to be popped to the front
Result: none

ScrollLayer
Description: scrolls a layer around in a bitmap
Library: layers.library
Offset: -$48
Syntax: ScrollLayer(dummy, layer, dx, dy)
C: VOID ScrollLayer(LONG, struct Layer *, LONG, LONG)
ML: ScrollLayer(a0, a1, d0, d1)
Arguments: dummy = unused
layer = the layer to scroll
dx = amount to scroll horizontally
dy = amount to scroll vertically
Result: none

ScrollRaster
Description: scrolls a rectangular area of a RastPort
Library: graphics.library
Offset: -$18C
Syntax: ScrollRaster(rastPort, dx, dy, x1, y1, x2, y2)
C: VOID ScrollRaster(struct RastPort *, WORD, WORD, WORD, WORD, WORD, WORD)
ML: ScrollRaster(a1, d0, d1, d2, d3, d4, d5)
Arguments: rastPort = RastPort structure
dx, dy = offset by which to scroll rectangle
x1, y1 = upper-left corner of rectangle
x2, y2 = lower-right corner of rectangle
Result: none

ScrollRasterBF (Revision 3.0)

Description: scrolls a rectangular area of a RastPort, clearing the vacated areas
Library: graphics.library
Offset: -$3EA
Syntax: ScrollRasterBF(rastPort, dx, dy, x1, y1, x2, y2)
C: VOID ScrollRasterBF(struct RastPort *, WORD, WORD, WORD, WORD, WORD, WORD)
ML: ScrollRasterBF(a1, d0, d1, d2, d3, d4, d5)
Arguments: rastPort = RastPort structure
dx, dy = offset by which to scroll rectangle
x1, y1 = upper-left corner of rectangle
x2, y2 = lower-right corner of rectangle
Result: none

ScrollVPort

Description: updates a ViewPort from a RasInfo structure
Library: graphics.library
Offset: -$24C
Syntax: ScrollVPort(viewPort)
C: VOID ScrollVPort(struct ViewPort *viewPort)
ML: ScrollVPort(a0)
Arguments: viewPort = ViewPort structure
Result: none

ScrollWindowRaster (Revision 3.0)

Description: provides an Intuition-friendly version of ScrollRasterBF( )
Library: intuition.library
Offset: -$31E
Syntax: ScrollWindowRaster(win, dx, dy, xmin, ymin, xmax, ymax)
C: VOID ScrollWindowRaster(struct Window *, WORD, WORD, WORD, WORD, WORD, WORD)
ML: ScrollWindowRaster(a1, d0, d1, d2, d3, d4, d5)
Arguments: win = window whose graphics are to scroll
dx, dy = distance and direction of scroll
xmin, ymin = upper left corner of bounding rectangle
xmax, ymax = lower right corner of bounding rectangle
Result: none

SDivMod32 (Revision 2.0)

Description: performs a signed 32- by 32-bit division and modulus
Library: utility.library
Offset: -$96
Syntax: quotient:remainder = SDivMod32(dividend, divisor)
C: LONG:LONG SDivMod32(LONG, LONG)
ML: d0/d1 = SDivMod32(d0, d1)

Arguments: dividend = signed 32-bit dividend
divisor = signed 32-bit divisor
Result: quotient = the signed 32-bit quotient of the division
remainder = the signed 32-bit remainder of the division

Seek

Description: moves the file pointer to a specified position in a file
Library: dos.library
Offset: -$42
Syntax: oldPosition = Seek(fh, position, mode)
C: LONG Seek(BPTR, LONG, LONG)
ML: d0 = Seek(d1, d2, d3)
Arguments: fh = filehandle of the file in question
position = amount by which to move file pointer
mode = relative position from which to move:
OFFSET_BEGINNING ($FFFFFFFF), OFFSET_CURRENT ($00000000),
OFFSET_END ($00000001)
Result: oldPosition = former file position; -1 if unsuccessful

SelectInput (Revision 2.0)

Description: selects a filehandle as the default input channel
Library: dos.library
Offset: -$126
Syntax: oldFh = SelectInput(fh)
C: BPTR SelectInput(BPTR)
ML: d0 = SelectInput(d1)
Arguments: fh = new default input filehandle
Result: oldFh = previous default input filehandle

SelectOutput (Revision 2.0)

Description: selects a filehandle as the default output channel
Library: dos.library
Offset: -$12C
Syntax: oldFh = SelectOutput(fh)
C: BPTR SelectOutput(BPTR)
ML: d0 = SelectOutput(d1)
Arguments: fh = new default output filehandle
Result: oldFh = previous default output filehandle

SendAmigaGuideCmdA (Revision 3.0)

Description: sends a command string to a system
Library: amigaguide.library
Offset: -$66
Syntax: success = SendAmigaGuideCmdA(handle, cmd, tags)

C: BOOL SendAmigaGuideCmdA(AMIGAGUIDECONTEXT, STRPTR, struct TagItem *)
ML: d0 = SendAmigaGuideCmd(a0, d0, d1)
Arguments: handle = handle to an AmigaGuide system
cmd = command to send—"ALINK <name>", "LINK <name>", "RX <macro>", "RXS <cmd>", "CLOSE", "QUIT"
tags = tag list of attributes—should be 0, but AGA_Context ($80000004) is defined
Result: success = zero if unsucessful

SendAmigaGuideContextA (Revision 3.0)
Description: aligns a system on the context ID
Library: amigaguide.library
Offset: -$60
Syntax: success = SendAmigaGuideContextA(handle, tags)
C: BOOL SendAmigaGuideContextA(AMIGAGUIDECONTEXT, struct TagItem *)
ML: d0 = SendAmigaGuideContextA(a0, d0)
Arguments: handle = handle to an AmigaGuide system
tags = tag list of attributes—should be zero
Result: success = zero if unsuccessful

SendIO
Description: initiates an I/O command
Library: exec.library
Offset: -$1CE
Syntax: SendIO(iORequest)
C: VOID SendIO(struct IORequest *)
ML: SendIO(a1)
Arguments: iORequest = an IORequest, or a device-specific extended IORequest
Result: none

SendPkt (Revision 2.0)
Description: sends a packet to a handler
Library: dos.library
Offset: -$F6
Syntax: SendPkt(packet, port, replyport)
C: VOID SendPkt(struct DosPacket *, struct MsgPort *, struct MsgPort *)
ML: SendPkt(d1, d2, d3)
Arguments: packet = packet to send
port = pr_MsgPort of handler process to receive the packet
replyport = MsgPort to receive the returned packet
Result: none

SetABPenDrMd (Revision 3.0)
Description: sets the pen colors and draw mode for a RastPort
Library: graphics.library

Offset: -$37E
Syntax: SetABPenDrMd(rastPort, apen, bpen, mode)
C: VOID SetABPenDrMd(struct RastPort *, ULONG, ULONG, ULONG)
ML: SetABPenDrMd(a1, d0, d1, d2)
Arguments: rastPort = RastPort structure
apen = primary pen value
bpen = secondary pen value
mode = draw mode; 0-255
Result: none

SetAmigaGuideAttrsA (Revision 3.0)

Description: sets an attribute
Library: amigaguide.library
Offset: -$6C
Syntax: retval = SetAmigaGuideAttrsA(handle, tags)
C: LONG SetAmigaGuideAttrsA(AMIGAGUIDECONTEXT, struct TagItem *)
ML: d0 = SetAmigaGuideAttrsA(a0, a1)
Arguments: handle = handle to an AmigaGuide system
tags = tag list of attribute pairs to set:
AGA_Activate ($80000003)
Result: retval = zero if unsuccessful

SetAmigaGuideContextA (Revision 3.0)

Description: sets the context ID
Library: amigaguide.library
Offset: -$5A
Syntax: success = SetAmigaGuideContextA(handle, index, tags)
C: BOOL SetAmigaGuideContextA(AMIGAGUIDECONTEXT, ULONG, struct TagItem *)
ML: d0 = SetAmigaGuideContextA(a0, d0, d1)
Arguments: handle = handle to an AmigaGuide system
index = index of node to display
tags = tag list of attributes—should be 0
Result: success = zero if index is invalid

SetAPen

Description: sets the primary pen for a RastPort
Library: graphics.library
Offset: -$156
Syntax: SetAPen(rastPort, pen)
C: VOID SetAPen(struct RastPort *, UBYTE)
ML: SetAPen(a1, d0)
Arguments: rastPort = RastPort structure
pen = pen color; 0-255
Result: none

SetArgStr (Revision 2.0)

Description: sets the arguments for the current process
Library: dos.library
Offset: -$21C
Syntax: oldptr = SetArgStr(ptr)
C: STRPTR SetArgStr(STRPTR)
ML: d0 = SetArgStr(d1)
Arguments: ptr = new argument string
Result: oldptr = previous argument string

SetAttrsA (Revision 2.0)

Description: specifies attribute values for an object
Library: intuition.library
Offset: -$288
Syntax: result = SetAttrsA(object, tags)
C: ULONG SetAttrs(APTR, struct TagItem *)
ML: d0 = SetAttrsA(a0, a1)
Arguments: object = abstract pointer to a boopsi object
tags = tag list of attributes:
see <intuition/classusr.h> for description
Result: result = depends on the object

SetBPen

Description: sets secondary pen for a RastPort
Library: graphics.library
Offset: -$15C
Syntax: SetBPen(rastPort, pen)
C: VOID SetBPen(struct RastPort *, UBYTE)
ML: SetBPen(a1, d0)
Arguments: rastPort = RastPort structure
pen = pen color; 0-255
Result: none

SetChipRev (Revision 3.0)

Description: attempts to activate a particular Amiga chipset
Library: graphics.library
Offset: -$378
Syntax: chiprevbits = SetChipRev(chipRev)
C: ULONG SetChipRev(ULONG)
ML: d0 = SetChipRev(d0)
Arguments: chipRev = chip revision that you would like to enable
Result: chiprevbits = actual bits set in GfxBase->ChipRevBits0

SetCollision

Description: sets user collision routine
Library: graphics.library
Offset: -$90
Syntax: SetCollision(num, routine, GInfo)
C: VOID SetCollision(ULONG, VOID (*)( ), struct GelsInfo *)
ML: SetCollision(d0, a0, a1)
Arguments: num = collision vector number
routine = user's collision routine
GInfo = GelsInfo structure
Result: none

SetComment

Description: changes a file's comment string
Library: dos.library
Offset: -$B4
Syntax: success = SetComment(name, comment)
C: BOOL SetComment(STRPTR, STRPTR)
ML: d0 = SetComment(d1, d2)
Arguments: name = filename of file to get new comment
comment = the new comment string
Result: success = zero if unsuccessful

SetConsoleTask (Revision 2.0)

Description: sets the default console for the process
Library: dos.library
Offset: -$204
Syntax: oldport = SetConsoleTask(port)
C: struct MsgPort *SetConsoleTask(struct MsgPort *)
ML: d0 = SetConsoleTask(d1)
Arguments: port = pr_MsgPort of the default console handler for the process
Result: oldport = previous ConsoleTask value

SetCurrentBinding

Description: sets an expansion board's private configuration data
Library: expansion.library
Offset: -$84
Syntax: SetCurrentBinding(currentBinding, size)
C: SetCurrentBinding(struct CurrentBinding *, ULONG)
ML: SetCurrentBinding(a0, d0)
Arguments: currentBinding = a CurrentBinding structure
size = number of bytes to copy
Result: none

SetCurrentDirName (Revision 2.0)

Description: sets the current directory name for the process
Library: dos.library
Offset: -$22E
Syntax: success = SetCurrentDirName(name)
C: BOOL SetCurrentDirName(STRPTR)
ML: d0 = SetCurrentDirName(d1)
Arguments: name = name of directory to be set
Result: success = zero if unsuccessful

SetCxObjPri (Revision 2.0)

Description: sets a commodity object's priority
Library: commodities.library
Offset: -$4E
Syntax: oldPri = SetCxObjPri(cxobj, pri)
C: LONG SetCxObjPri(CxObj *, LONG)
ML: d0 = SetCxObjPri(a0, d0)
Arguments: cxobj = commodity object to modify
pri = the object's new priority; -128 to +127
zero is normal
Result: oldPri = previous priority of the object

SetDefaultPubScreen (Revision 2.0)

Description: makes a public screen the default public screen
Library: intuition.library
Offset: -$21C
Syntax: SetDefaultPubScreen(name)
C: VOID SetDefaultPubScreen(char *)
ML: SetDefaultPubScreen(a0)
Arguments: name = name of public screen; 0 to indicate the Workbench (or the default public screen)
Result: none

SetDMRequest

Description: attaches a DMRequester
Library: intuition.library
Offset: -$102
Syntax: success = SetDMRequest(window, dmRequest)
C: BOOL SetDMRequest(struct Window *, struct Requester *)
ML: d0 = SetDMRequest(a0, a1)
Arguments: window = window to receive the requester
dmRequest = requester to attatch
Result: success = zero if unsuccessful

SetDrMd

Description: sets the drawing mode for a RastPort
Library: graphics.library
Offset: -$162
Syntax: SetDrMd(rastPort, mode)
C: VOID SetDrMd(struct RastPort *, UBYTE)
ML: SetDrMd(a1, d0)
Arguments: rastPort = RastPort structure
mode = drawing mode; 0-255
Result: none

SetDTAttrsA (Revision 3.0)

Description: sets a DataType object's attributes
Library: datatypes.library
Offset: -$3C
Syntax: retval = SetDTAttrsA(object, window, requester, tags)
C: ULONG SetDTAttrsA(Object *, struct Window *, struct Requester *, struct TagItem *)
ML: d0 = SetDTAttrsA(a0, a1, a2, a3)
Arguments: object = Object structure returned by NewDTObjectA( )
window = window to which the object is attached
requester = requester to which the object is attached
tags = tag list specifying attributes to set—see <libraries/datatypeclass.h> for description
Result: retval = zero if unsuccessful

SetEditHook (Revision 2.0)

Description: modifies the operation of a string gadget
Library: intuition.library
Offset: -$1EC
Syntax: oldHook = SetEditHook(hook)
C: struct Hook *SetEditHook(struct Hook *)
ML: d0 = SetEditHook(a0)
Arguments: hook = Hook structure specifying function to call for each key pressed
Result: oldHook = previous string gadget hook, if it existed
WARNING! This function is unsupported and should not be used in commercial applications until Commodore sanctions such use.

SetExcept

Description: defines certain signals to cause exceptions
Library: exec.library
Offset: -$138
Syntax: oldSignals = SetExcept(newSignals, signalMask)
C: ULONG SetExcept(ULONG, ULONG)
ML: d0 = SetExcept(d0, d1)

Arguments: newSignals = the new values for the signals specified in signalMask
signalMask = the set of signals to be affected
Result: oldSignals = the prior exception signals

SetFileDate (Revision 2.0)
Description: sets the modification date for a file or directory
Library: dos.library
Offset: -$18C
Syntax: success = SetFileDate(name, date)
C: BOOL SetFileDate(STRPTR, struct DateStamp *)
ML: d0 = SetFileDate(d1, d2)
Arguments: name = name of file or directory
date = new modification date
Result: success = zero if unsuccessful

SetFileSize (Revision 2.0)
Description: sets the size of a file
Library: dos.library
Offset: -$1C8
Syntax: newsize = SetFileSize(fh, offset, mode)
C: LONG SetFileSize(BPTR, LONG, LONG)
ML: d0 = SetFileSize(d1, d2, d3)
Arguments: fh = filehandle of file to be truncated or expanded
offset = new end-of-file position; specified as an offset
from one of the following positions
mode = relative position from which to move:
OFFSET_BEGINNING ($FFFFFFFF), OFFSET_CURRENT ($00000000),
OFFSET_END ($00000001)
Result: newsize = new file size; -1 if unsuccessful

SetFileSysTask (Revision 2.0)
Description: sets the default filesystem for the current process
Library: dos.library
Offset: -$210
Syntax: oldport = SetFileSysTask(port)
C: struct MsgPort *SetFileSysTask(struct MsgPort *)
ML: d0 = SetFileSysTask(d1)
Arguments: port = pr_MsgPort of the default filesystem for the process
Result: oldport = previous FileSysTask value

SetFilter (Revision 2.0)
Description: changes a commodity filter
Library: commodities.library
Offset: -$78
Syntax: SetFilter(filter,text)

C: VOID SetFilter(CxObj *, STRPTR)
ML: SetFilter(a0, a1)
Arguments: filter = filter object to modify
text = new conditions to match
Result: none

SetFilterIX (Revision 2.0)
Description: changes a commodity filter
Library: commodities.library
Offset: -$7E
Syntax: SetFilterIX(filter,ix)
C: VOID SetFilterIX(CxObj *, IX *)
ML: SetFilterIX(a0, a1)
Arguments: filter = filter object to modify
ix = new conditions to match
Result: none

SetFont
Description: sets the font for a RastPort
Library: graphics.library
Offset: -$42
Syntax: SetFont(rastPort, font)
C: VOID SetFont(struct RastPort *, struct TextFont *)
ML: SetFont(a1, a0)
Arguments: rastPort = RastPort structure
font = TextFont structure as returned by OpenFont( ) or OpenDiskFont( )
Result: none

SetFunction
Description: changes a function vector in a library
Library: exec.library
Offset: -$1A4
Syntax: oldFunc = SetFunction(library, funcOffset, funcEntry)
C: APTR SetFunction(struct Library *, LONG, APTR)
ML: d0 = SetFunction(a1, a0, d0)
Arguments: library = the library to change
funcOffset = the offset of the function to replace
funcEntry = the new function
Result: oldFunc = the old function that was just replaced

SetGadgetAttrsA
Description: specifies attribute values for a boopsi gadget
Library: intuition.library
Offset: -$294
Syntax: result = SetGadgetAttrsA(gadget, window, requester, tags)

C: ULONG SetGadgetAttrs(struct Gadget *, struct Window *, struct Requester *, struct TagItem *)
ML: d0 = SetGadgetAttrsA(a0, a1, a2, a3)
Arguments: gadget = abstract pointer to a boopsi gadget
window = window containing gadget
requester = requester containing gadget; may be 0
tags = tag list of attributes:
see <intuition/classusr.h> for description
Result: result = nonzero if gadget needs refreshing

SetInfoA (Revision 3.0)

Description: sets font metrics
Libra bullet.library
Offset: -$2A
Syntax: error = SetInfoA(handle, tags)
C: ULONG SetInfo(struct GlyphEngine *, struct TagItem *)
ML: d0 = SetInfoA(a0, a1)
Arguments: handle = handle returned by OpenEngine( )
tags = tag list specifying data to modify—see <diskfont/diskfonttags.h> for description
Result: error = zero if successful; nonzero error-code if unsuccessful

SetIntVector

Description: sets a new handler for a system interrupt vector
Library: exec.library
Offset: -$A2
Syntax: oldInterrupt = SetIntVector(intNumber, interrupt)
C: struct Interrupt *SetIntVector(ULONG, struct Interrupt *)
ML: d0 = SetIntVector(d0, a1)
Arguments: intNum = the Paula interrupt bit number; 0-14
interrupt = an Interrupt structure containing the handler's entry point and data segment pointer; 0 will remove the current interrupt and set illegal values for IS_CODE and IS_DATA
Result: oldInterrupt = a pointer to the old interrupt structure

SetIoErr (Revision 2.0)

Description: sets the value returned by IoErr( )
Library: dos.library
Offset: -$1CE
Syntax: oldcode = SetIoErr(code)
C: LONG SetIoErr(LONG)
ML: d0 = SetIoErr(d1)
Arguments: code = code to be returned by a call to IoErr( )
Result: oldcode = previous pending error code

SetKeyMapDefault (Revision 2.0)

Description: sets the current default key map
Library: keymap.library
Offset: -$1E
Syntax: SetKeyMapDefault(keyMap)
C: VOID SetKeyMapDefault(struct KeyMap *)
ML: SetKeyMapDefault(a0)
Arguments: keyMap = a permanently allocated KeyMap structure
Result: none

SetLocalItemPurge (Revision 2.0)

Description: sets purge vector for a local context item
Library: iffparse.library
Offset: -$C6
Syntax: SetLocalItemPurge(localItem, purgeHook)
C: VOID SetLocalItemPurge(struct LocalContextItem *, struct Hook *)
ML: SetLocalItemPurge(a0, a1)
Arguments: localItem = a local context item
purgeHook = a Hook structure
Result: none

SetMaxPen (Revision 3.0)

Description: sets maximum pen value for a rastport
Library: graphics.library
Offset: -$3DE
Syntax: SetMaxPen(rastPort, maxpen)
C: VOID SetMaxPen(struct RastPort *, ULONG)
ML: SetMaxPen (a0, d0)
Arguments: rastPort = RastPort structure
maxpen = maximum pen value
Result: none

SetMenuStrip

Description: attaches a menu strip to a window
Library: intuition.library
Offset: -$108
Syntax: success = SetMenuStrip(window, menu)
C: BOOL SetMenuStrip(struct Window *, struct Menu *)
ML: d0 = SetMenuStrip(a0, a1)
Arguments: window = window to receive menu strip
menu = first menu in menu strip
Result: always nonzero

SetMode (Revision 2.0)

Description: determines the behavior of a handler
Library: dos.library
Offset: -$1AA
Syntax: success = SetMode(fh, mode)
C: BOOL SetMode(BPTR, LONG)
ML: d0 = SetMode(d1, d2)
Arguments: fh = filehandle of handler you wish to affect
mode = new mode; for CON: handlers, use 0 for normal CON: mode; 1 for RAW mode
Result: success = zero if unsuccessful

SetMouseQueue (Revision 2.0)

Description: changes the limit on pending mouse messages
Library: intuition.library
Offset: -$1F2
Syntax: oldQueueLength = SetMouseQueue(window, queueLength)
C: LONG SetMouseQueue(struct Window *, UWORD)
ML: d0 = SetMouseQueue(a0, d0)
Arguments: window = window with mouse messages
queueLength = maximum number of unacknowledged mouse messages to track
Result: oldQueueLength = previous mouse-message limit;
1 if unsuccessful

SetOutlinePen (Revision 3.0)

Description: sets the outline pen value for a RastPort
Library: graphics.library
Offset: -$3D2
Syntax: old_pen = SetOutlinePen(rastPort, pen)
C: ULONG SetOutlinePen(struct RastPort *, ULONG)
ML: d0 = SetOutlinePen (a0, d0)
Arguments: rastPort = RastPort structure
pen = pen number
Result: none

SetOwner (Revision 3.0)

Description: sets owner information for a file or directory
Library: dos.library
Offset: -$3DE
Syntax: success = SetOwner(name, ownerInfo)
C: BOOL SetOwner (STRPTR, LONG)
ML: d0 = SetOwner(d1, d2)
Arguments: name = name of file or directory
ownerInfo = group id (bits 0-15) and owner uid (bits 16-31)
Result: success = zero if unsuccessful

SetPointer

Description: changes the shape of a window's mouse pointer
Library: intuition.library
Offset: -$10E
Syntax: SetPointer(window, pointer, height, width, xOffset, yOffset)
C: VOID SetPointer(struct Window *, UWORD *, WORD, WORD, WORD, WORD)
ML: SetPointer(a0, a1, d0, d1, d2, d3)
Arguments: window = window to receive new pointer
pointer = sprite data defining new shape
height = pointer height
width = pointer width; must be <= 16
xOffset = horizontal offset of hotspot
yOffset = vertical offset of hotspot
Result: none

SetPrefs

Description: changes Preferences' current settings
Library: intuition.library
Offset: -$144
Syntax: prefs = SetPrefs(prefBuffer, size, inform)
C: struct Preferences *SetPrefs(struct Preferences *, LONG, BOOL)
ML: d0 = SetPrefs(a0, d0, d1)
Arguments: prefBuffer = buffer containing the new Preferences settings
size = size of buffer
inform = nonzero to send a NEWPREFS message to all interested windows and tasks
Result: prefs = pointer to prefBuffer

SetProgramDir (Revision 2.0)

Description: sets the directory returned by GetProgramDir
Library: dos.library
Offset: -$252
Syntax: oldlock = SetProgramDir(lock)
C: BPTR SetProgramDir(BPTR)
ML: d0 = SetProgramDir(d1)
Arguments: lock = lock on the directory from which the current program was loaded
Result: oldlock = previous ProgramDir

SetProgramName (Revision 2.0)

Description: sets the name of the program being run
Library: dos.library
Offset: -$23A
Syntax: success = SetProgramName(name)
C: BOOL SetProgramName(STRPTR)
ML: d0 = SetProgramName(d1)

Arguments: name = program name
Result: success = zero if unsuccessful

SetPrompt (Revision 2.0)
Description: changes the CLI/shell prompt
Library: dos.library
Offset: -$246
Syntax: success = SetPrompt(name)
C: BOOL SetPrompt(STRPTR)
ML: d0 = SetPrompt(d1)
Arguments: name = new prompt string
Result: success = zero if unsuccessful

SetProtection
Description: sets the protection flags for a file or directory
Library: dos.library
Offset: -$BA
Syntax: success = SetProtection(name, mask)
C: BOOL SetProtection(STRPTR, LONG)
ML: d0 = SetProtection(d1, d2)
Arguments: name = name of file or directory
mask = file protection bits—FIBF_DELETE ($00000001), FIBF_EXECUTE ($00000002), FIBF_WRITE ($00000004), FIBF_READ ($00000008), FIBF_ARCHIVE ($00000010), FIBF_PURE ($00000020), FIBF_SCRIPT ($00000040)
Result: success = zero if unsuccessful

SetPubScreenModes (Revision 2.0)
Description: changes the behavior of all public screens
Library: intuition.library
Offset: -$222
Syntax: oldModes = SetPubScreenModes(modes)
C: UWORD SetPubScreenModes(UWORD)
ML: d0 = SetPubScreenModes(d0)
Arguments: modes = new global modes flags
SHANGHAI ($0001), POPPUBSCREEN ($0002)
Result: oldModes = previous global mode settings

SetRast
Description: sets an entire drawing area to a specified color
Library: graphics.library
Offset: -$EA
Syntax: SetRast(rastPort, pen)
C: VOID SetRast(struct RastPort *, UBYTE)
ML: SetRast(a1, d0)

Arguments: rastPort = RastPort structure to fill
pen = pen number; 0-255
Result: none

SetRGB32 (Revision 3.0)
Description: sets one color in a ViewPort
Library: graphics.library
Offset: -$354
Syntax: SetRGB4(viewPort, pen, red, green, blue)
C: VOID SetRGB4(struct ViewPort *, WORD, ULONG, ULONG, ULONG)
ML: SetRGB4(a0, d0, d1, d2, d3)
Arguments: viewPort = ViewPort structure
pen = pen number
red, green, blue = color attributes
Result: none

SetRGB32CM (Revision 3.0)
Description: sets one color in a ColorMap
Library: graphics.library
Offset: -$3E4
Syntax: SetRGB4CM(colorMap, pen, red, green, blue)
C: VOID SetRGB4CM(struct ColorMap *, WORD, ULONG, ULONG, ULONG)
ML: SetRGB4CM(a0, d0, d1, d2, d3)
Arguments: colorMap = ColorMap structure
pen = pen number
red, green, blue = color attributes
Result: none

SetRGB4
Description: sets one color in a ViewPort
Library: graphics.library
Offset: -$120
Syntax: SetRGB4(viewPort, pen, red, green, blue)
C: VOID SetRGB4(struct ViewPort *, WORD, UBYTE, UBYTE, UBYTE)
ML: SetRGB4(a0, d0, d1, d2, d3)
Arguments: viewPort = ViewPort structure
pen = pen number
red, green, blue = color attributes
Result: none

SetRGB4CM
Description: sets one color in a ColorMap
Library: graphics.library
Offset: -$276
Syntax: SetRGB4CM(colorMap, pen, red, green, blue)

C: VOID SetRGB4CM(struct colorMap *, WORD, UBYTE, UBYTE, UBYTE)
ML: SetRGB4CM(a0, d0, d1, d2, d3)
Arguments: colorMap = ColorMap structure
pen = pen number
red, green, blue = color attributes
Result: none

SetRPAttrA (Revision 3.0)
Description: modifies a RastPort's settings via a tag list
Library: graphics.library
Offset: -$40E
Syntax: SetRPAttrA(rastPort,tags)
C: VOID SetRPAttrA(struct RastPort *, struct TagItem *)
ML: SetRPAttrA(a0, a1)
Arguments: rastPort = RastPort to modify
tags = tag list of attributes:
RPTAG_Font ($80000000), RPTAG_APen ($80000002), RPTAG_BPen ($80000003), RPTAG_DrMd ($80000004), RPTAG_OutLinePen ($80000005), RPTAG_WriteMask ($80000006), RPTAG_MaxPen ($80000007), RPTAG_DrawBounds ($80000008)
Result: none

SetSignal
Description: defines the state of this task's signals
Library: exec.library
Offset: -$132
Syntax: oldSignals = SetSignal(newSignals, signalMask)
C: ULONG SetSignal(ULONG, ULONG)
ML: d0 = SetSignal(d0, d1)
Arguments: newSignals = the new values for the signals specified in signalSet
signalMask = the set of signals to be affected
Result: oldSignals = the prior values for all signals

SetSoftStyle
Description: sets the soft style of the current font
Library: graphics.library
Offset: -$5A
Syntax: newStyle = SetSoftStyle(rastPort, style, enable)
C: ULONG SetSoftStyle(struct RastPort *, ULONG, ULONG)
ML: d0 = SetSoftStyle(a1, d0, d1)
Arguments: rastPort = RastPort structure
style = new font style bits
enable = mask that determines which style bits can be changed
Result: newStyle = resulting soft style

SetSR

Description: gets and/or sets the processor status register
Library: exec.library
Offset: -$90
Syntax: oldSR = SetSR(newSR, mask)
C: ULONG SetSR(ULONG, ULONG)
ML: d0 = SetSR(d0, d1)
Arguments: newSR = new values for bits specified in the mask
mask = bits to be changed
Result: oldSR = the status register prior to being changed

SetTaskPri

Description: gets and sets the priority of a task
Library: exec.library
Offset: -$12C
Syntax: oldPriority = SetTaskPri(task, priority)
C: BYTE SetTaskPri(struct Task *, LONG)
ML: d0 = SetTaskPri(a1, d0)
Arguments: task = task to be affected
priority = the new priority for the task
Result: oldPriority = the task's previous priority

SetTranslate (Revision 2.0)

Description: replaces a translator object's translation list
Library: commodities.library
Offset: -$72
Syntax: SetTranslate(translator,events)
C: VOID SetTranslate(CxObj *, struct InputEvent *)
ML: SetTranslate(a0, a1)
Arguments: translator = translator object to modify
events = new input event translation list
Result: none

SetVar (Revision 2.0)

Description: sets an environment variable
Library: dos.library
Offset: -$384
Syntax: success = SetVar(name, buffer, size, flags)
C: BOOL SetVar(STRPTR, STRPTR, LONG, ULONG)
ML: d0 = SetVar(d1, d2, d3, d4)
Arguments: name = name of variable
buffer = value to assign to variable
size = size of buffer;
a buffer size of 1 indicates that your buffer contains a NULL-terminated string

flags = type of variable:
GVF_LOCAL_ONLY ($00000100), GVF_GLOBAL_ONLY ($00000200)
Result: success = zero if unsuccessful

SetVBuf (Revision 2.0)
Description: sets I/O buffering mode to be used for a filehandle
Library: dos.library
Offset: -$16E
Syntax: error = SetVBuf(fh, buff, type, size)
C: LONG SetVBuf(BPTR, STRPTR, LONG, LONG)
ML: d0 = SetVBuf(d1, d2, d3, d4)
Arguments: fh = filehandle to affect
buff = buffer to use for buffered I/O; 0 if you don't want buffered I/O
type = buffering mode
BUF_LINE ($00000000), BUF_FUL ($00000001), BUF_NONE ($00000002)
size = size of buffer; sizes less than 208 are ignored
Result: error = nonzero if unsuccessful

SetWindowPointerA (Revision 3.0)
Description: defines a window's mouse pointer shape
Library: intuition.library
Offset: -$330
Syntax: SetWindowPointerA(window, tags)
C: VOID SetWindowPointer(struct Window *, struct TagItem *)
ML: SetWindowPointerA(a0, a1)
Arguments: window = window to receive new mouse pointer
tags = tag list of attributes—see <intuition/intuition.h> for description
Result: none

SetWindowTitles
Description: sets the titles for both a window and its screen
Library: intuition.library
Offset: -$114
Syntax: SetWindowTitles(window, windowTitle, screenTitle)
C: VOID SetWindowTitles(struct Window *, UBYTE *, UBYTE *)
ML: SetWindowTitles(a0, a1, a2)
Arguments: window = window to receive new name
windowTitle = new window title; -1 to leave unchanged
screenTitle = new screen title; -1 to leave unchanged
Result: none

SetWriteMask (Revision 3.0)
Description: sets a RastPort's pixel write mask
Library: graphics.library
Offset: -$3D8

Syntax: success = SetWriteMask(rastPort, mask)
C: ULONG SetWriteMask(struct RastPort *, ULONG)
ML: d0 = SetWriteMask (a0, d0)
Arguments: rastPort = RastPort structure
mask = a longword mask value
Result: success = zero if unsuccessful

ShowTitle

Description: determines if a screen title bar or backdrop window appears topmost
Library: intuition.library
Offset: -$11A
Syntax: ShowTitle(screen, showIt)
C: VOID ShowTitle(struct Screen *, BOOL)
ML: ShowTitle(a0, d0)
Arguments: screen = screen with backdrop window
showIt = nonzero to show the title bar; zero to hide title bar behind backdrop window
Result: none

Signal

Description: signals a task
Library: exec.library
Offset: -$132
Syntax: Signal(task, signals)
C: VOID Signal(struct Task *,ULONG)
ML: Signal(a1, d0)
Arguments: task = the task to be signalled
signals = the signals to be sent
Result: none

SizeLayer

Description: sizes a nonbackdrop layer
Library: layers.library
Offset: -$42
Syntax: success = SizeLayer(dummy, layer, dx, dy)
C: LONG SizeLayer(LONG, struct Layer *, LONG, LONG)
ML: d0 = SizeLayer(a0, a1, d0, d1)
Arguments: dummy = unused
layer = the layer to size; must not be a backdrop layer
dx = amount to change the current x size
dy = amount to change the current y size
Result: success = zero if unsuccessful

SizeWindow

Description: resizes a window
Library: intuition.library

Offset: -$120
Syntax: SizeWindow(window, deltaX, deltaY)
C: VOID SizeWindow(struct Window *, WORD, WORD)
ML: SizeWindow(a0, d0, d1)
Arguments: window = window to resize
deltaX = change in window's width; negative values make window thinner
deltaY = change in window's height; negative values make window shorter
Result: none

SMult32 (Revision 2.0)
Description: performs a signed 32- by 32-bit multiplication with 32-bit result
Library: utility.library
Offset: -$8A
Syntax: result = SMult32(arg1, arg2)
C: LONG SMult32(LONG, LONG)
ML: d0 = SMult32(d0, d1)
Arguments: arg1 = signed 32-bit multiplicand
arg2 = signed 32-bit multiplier
Result: result = the signed 32-bit product of arg1 and arg2

SMult64 (Revision 3.0)
Description: performs signed 32- by 32-bit multiplication with 64-bit result
Library: utility.library
Offset: -$C6
Syntax: result = SMult64(arg1, arg2)
C: LONG SMult64(LONG, LONG)
ML: d0 = SMult64(d0, d1)
Arguments: arg1 = signed 32-bit multiplicand
arg2 = signed 32-bit multiplier
Result: result = the signed 64-bit product of arg1 and arg2

SortGList
Description: sorts the current gel list by screen position
Library: graphics.library
Offset: -$96
Syntax: SortGList(rastPort)
C: VOID SortGList(struct RastPort *)
ML: SortGList(a1)
Arguments: rastPort = RastPort structure containing the gel list
Result: none

SPAbs
Description: computes the absolute value of an FFP number
Library: mathffp.library
Offset: -$36

Syntax: fnum2 = SPAbs(fnum1)
C: float SPAbs(float)
ML: d0 = SPAbs(d0)
Arguments: fnum1 = number for which to compute absolute value
Result: fnum2 = absolute value of fnum1

SPAcos
Description: computes the arccosine of an FFP number
Library: mathtrans.library
Offset: -$78
Syntax: fnum2 = SPAcos(fnum1)
C: float SPAcos(float)
ML: d0 = SPAcos(d0)
Arguments: fnum1 = number for which to compute arccosine
Result: fnum2 = arccosine of fnum1

SPAdd
Description: computes the sum of two FFP numbers
Library: mathffp.library
Offset: -$42
Syntax: fnum3 = SPAdd(fnum1, fnum2)
C: float SPAdd(float, float)
ML: d0 = SPAdd(d1, d0)
Arguments: fnum1, fnum2 = summands
Result: fnum3 = value of fnum1 + fnum2.

SPAsin
Description: computes the arcsine of an FFP number
Library: mathtrans.library
Offset: -$72
Syntax: fnum2 = SPAsin(fnum1)
C: float SPAsin(float)
ML: d0 = SPAsin(d0)
Arguments: fnum1 = number for which to compute arcsine
Result: fnum2 = arcsine of fnum1

SPAtan
Description: computes the arctangent of an FFP number
Library: mathtrans.library
Offset: -$1E
Syntax: fnum2 = SPAtan(fnum1)
C: float SPAtan(float)
ML: d0 = SPAtan(d0)
Arguments: fnum1 = number for which to compute arctangent
Result: fnum2 = arctangent of fnum1

SPCeil

Description: computes the ceiling of an FFP number
Library: mathffp.library
Offset: -$60
Syntax: fnum2 = SPCeil(fnum1)
C: float SPCeil(float)
ML: d0 = SPCeil(d0)
Arguments: fnum1 = number for which to compute ceiling
Result: fnum2 = smallest whole number greater than or equal to fnum1

SPCmp

Description: compares two FFP numbers and sets condition codes
Library: mathffp.library
Offset: -$2A
Syntax: result = SPCmp(fnum1, fnum2)
C: int SPCmp(float, float)
ML: d0 = SPCmp(d0, d1)
Arguments: fnum1, fnum2 = numbers to compare
Result: result = 1 if fnum1 > fnum2, zero if fnum1 = fnum2, -1 if fnum1 < fnum2

SPCos

Description: computes the cosine of an FFP number
Library: mathtrans.library
Offset: -$2A
Syntax: fnum2 = SPCos(fnum1)
C: float SPCos(float)
ML: d0 = SPCos(d0)
Arguments: fnum1 = number for which to compute cosine
Result: fnum2 = cosine of fnum1

SPCosh

Description: computes the hyperbolic cosine of an FFP number
Library: mathtrans.library
Offset: -$42
Syntax: fnum2 = SPCosh(fnum1)
C: float SPCosh(float)
ML: d0 = SPCosh(d0)
Arguments: fnum1 = number for which to compute hyperbolic cosine
Result: fnum2 = hyperbolic cosine of fnum1

SPDiv

Description: computes the quotient of two FFP numbers
Library: mathffp.library
Offset: -$54
Syntax: fnum3 = SPDiv(fnum1, fnum2)

C: float SPDiv(float, float)
ML: d0 = SPDiv(d1, d0)
Arguments: fnum1 = dividend
fnum2 = divisor
Result: fnum3 = value of fnum1 / fnum2

SPExp
Description: computes the exponential of an FFP number
Library: mathtrans.library
Offset: -$4E
Syntax: fnum2 = SPExp(fnum1)
C: float SPExp(float)
ML: d0 = SPExp(d0)
Arguments: fnum1 = number for which to compute the exponential
Result: fnum2 = exponential of fnum1

SPFieee
Description: converts a IEEE single precision number to an FFP number
Library: mathtrans.library
Offset: -$6C
Syntax: fnum = SPFieee(ieeenum)
C: float SPFieee(float)
ML: d0 = SPFieee(d0)
Arguments: ieeenum = IEEE single precision number to convert
Result: fnum = FFP representation of ieeenum

SPFix
Description: converts an FFP number to an integer
Library: mathffp.library
Offset: -$1E
Syntax: inum = SPFix(fnum)
C: int SPFix(float)
ML: d0 = SPFix(d0)
Arguments: fnum = number to convert
Result: inum = truncated integer value of fnum

SPFloor
Description: computes the floor of a FFP number
Library: mathffp.library
Offset: -$5A
Syntax: fnum2 = SPFloor(fnum1)
C: float SPFloor(float)
ML: d0 = SPFloor(d0)
Arguments: fnum1 = number for which to compute floor
Result: fnum2 = largest whole number less than or equal to fnum1

SPFlt

Description: converts an integer to an FFP number
Library: mathffp.library
Offset: -$24
Syntax: fnum = SPFlt(inum)
C: float SPFlt(int)
ML: d0 = SPFlt(d0)
Arguments: inum = integer to convert
Result: fnum = floating point representation of inum

SplitName (Revision 2.0)

Description: splits pathnames to isolate a filename, directory name, or device name
Library: dos.library
Offset: -$19E
Syntax: newpos = SplitName(name, separator, buf, oldpos, size)
C: WORD SplitName(STRPTR, UBYTE, STRPTR, WORD, LONG)
ML: d0 = SplitName(d1, d2, d3, d4, d5)
Arguments: name = pathname to be split
separator = character where split should occur (usually a : or /)
buf = buffer to receive separated name
oldpos = offset into pathname
size = size of bufer
Result: newpos = new offset position for next call to SplitName( );
-1 if pathname could not be split

SPLog

Description: computes the natural logarithm of an FFP number
Library: mathtrans.library
Offset: -$54
Syntax: fnum2 = SPLog(fnum1)
C: float SPLog(float)
ML: d0 = SPLog(d0)
Arguments: fnum1 = number for which to compute natural logarithm
Result: fnum2 = natural logarithm of fnum1

SPLog10

Description: computes the base 10 logarithm of an FFP number
Library: mathtrans.library
Offset: -$7E
Syntax: fnum2 = SPLog10(fnum1)
C: float SPLog10(float)
ML: d0 = SPLog10(d0)
Arguments: fnum1 = number for which to compute logarithm
Result: fnum2 = base 10 logarithm of fnum1

SPMul

Description: computes the product of two FFP numbers
Library: mathffp.library
Offset: -$4E
Syntax: fnum3 = SPMul(fnum1, fnum2)
C: float SPMul(float, float)
ML: d0 = SPMul(d1, d0)
Arguments: fnum1 = multiplicand
fnum2 = multiplier
Result: fnum3 = value of fnum1 * fnum2

SPNeg

Description: negates an FFP number
Library: mathffp.library
Offset: -$3C
Syntax: fnum2 = SPNeg(fnum1)
C: float SPNeg(float)
ML: d0 = SPNeg(d0)
Arguments: fnum1 = number to negate
Result: fnum2 = -fnum1

SPPow

Description: raises a number to a power
Library: mathtrans.library
Offset: -$5A
Syntax: result = SPPow(fnum1, fnum2)
C: float = SPPow(float, float)
ML: d0 = SPPow(d1, d0)
Arguments: fnum1 = exponent to fnum2
fnum2 = value to raise to the power fnum1
Result: result = value of fnum2 raised to the fnum1 power

SPSin

Description: computes the sine of an FFP number
Library: mathtrans.library
Offset: -$24
Syntax: fnum2 = SPSin(fnum1)
C: float SPSin(float)
ML: d0 = SPSin(d0)
Arguments: fnum1 = number for which to compute sine
Result: fnum2 = sine of fnum1

SPSincos

Description: computes sine and cosine of an FFP number
Library: mathtrans.library

Offset: -$36
Syntax: fnum3 = SPSincos(pfnum2, fnum1)
C: float = SPSincos(float *, float)
ML: d0 = SPSincos(d1, d0)
Arguments: fnum1 = number for which to compute sine and cosine
pfnum2 = buffer to receive the cosine of fnum1
Result: fnum3 = sine of fnum1

SPSinh

Description: computes the hyperbolic sine of an FFP number
Library: mathtrans.library
Offset: -$3C
Syntax: fnum2 = SPSinh(fnum1)
C: float SPSinh(float)
ML: d0 = SPSinh(d0)
Arguments: fnum1 = number for which to compute hyperbolic sine
Result: fnum2 = hyperbolic sine of fnum1

SPSqrt

Description: computes the square root of an FFP number
Library: mathtrans.library
Offset: -$60
Syntax: fnum2 = SPSqrt(fnum1)
C: float SPSqrt(float)
ML: d0 = SPSqrt(d0)
Arguments: fnum1 = number for which to compute square root
Result: fnum2 = square root of fnum1

SPSub

Description: computes the difference between two FFP numbers
Library: mathffp.library
Offset: -$48
Syntax: fnum3 = SPSub(fnum1, fnum2)
C: float SPSub(float, float)
ML: d0 = SPSub(d1, d0)
Arguments: fnum1 = minuend
fnum2 = subtrahend
Result: fnum3 = value of fnum1 - fnum2

SPTan

Description: computes the tangent of an FFP number
Library: mathtrans.library
Offset: -$30
Syntax: fnum2 = SPTan(fnum1)
C: float SPTan(float)

ML: d0 = SPTan(d0)
Arguments: fnum1 = number for which to compute tangent
Result: fnum2 = tangent of fnum1

SPTanh
Description: computes the hyperbolic tangent of an FFP number
Library: mathtrans.library
Offset: -$48
Syntax: fnum2 = SPTanh(fnum1)
C: float SPTanh(float)
ML: d0 = SPTanh(d0)
Arguments: fnum1 = number for which to compute hyperbolic tangent
Result: fnum2 = hyperbolic tangent of fnum1

SPTieee
Description: converts an FFP number to an IEEE single precision number
Library: mathtrans.library
Offset: -$66
Syntax: ieeenum = SPTieee(fnum)
C: float SPTieee(float)
ML: d0 = SPTieee(d0)
Arguments: fnum = FFP number to convert
Result: ieeenum = IEEE single precision representation of fnum

SPTst
Description: compares an FFP number to zero (0.0)
Library: mathffp.library
Offset: -$30
Syntax: result = SPTst(fnum)
C: int SPTst(float)
ML: d0 = SPTst(d1)
Arguments: fnum = number to compare to zero
Result: result = 1 if fnum1 > 0.0, zero if fnum1 = 0.0, -1 if fnum1 < 0.0

StackSwap (Revision 2.0)
Description: replaces a task's stack
Library: exec.library
Offset: -$2DC
Syntax: StackSwap(newStack)
C: VOID StackSwap(struct StackSwapStruct *)
ML: StackSwap(a0)
Arguments: newStack = defines the new upper and lower bounds
Result: newStack = contains old values prior to change

StartNotify (Revision 2.0)

Description: sends the file system a notification request for a file or directory
Library: dos.library
Offset: -$378
Syntax: success = StartNotify(notifystructure)
C: BOOL StartNotify(struct NotifyRequest *)
ML: d0 = StartNotify(d1)
Arguments: notifystructure = an initialized NotifyRequest structure
Result: success = zero if unsuccessful

StopChunk (Revision 2.0)

Description: declares a chunk which should cause ParseIFF( ) to return
Library: iffparse.library
Offset: -$7E
Syntax: error = StopChunk(iff, type, id)
C: LONG StopChunk(struct IFFHandle *, LONG, LONG)
ML: d0 = StopChunk(a0, d0, d1)
Arguments: iff = an IFFHandle structure
type = type code for chunk
id = identifier for chunk
Result: error = zero if successful; nonzero error-code if unsuccessful

StopChunks (Revision 2.0)

Description: declares multiple stop chunks
Library: iffparse.library
Offset: -$84
Syntax: error = StopChunks(iff, propArray, numPairs)
C: LONG StopChunks(struct IFFHandle *, LONG *, LONG)
ML: d0 = StopChunks(a0, a1, d0)
Arguments: iff = pointer to an IFFHandle structure
propArray = array of longword chunk types and identifiers
numPairs = number of pairs in the array
Result: error = zero if successful; nonzero error-code if unsuccessful

StopOnExit (Revision 2.0)

Description: declares a stop condition for exiting a chunk
Library: iffparse.library
Offset: -$96
Syntax: error = StopOnExit(iff, type, id)
C: LONG StopOnExit(struct IFFHandle *, LONG, LONG)
ML: d0 = StopOnExit(a0, d0, d1)

Arguments: iff = an IFFHandle structure
type = type code for chunk
id = identifier for chunk
Result: error = zero if successful; nonzero error-code if unsuccessful

StoreItemInContext (Revision 2.0)

Description: stores local context item in given context node
Library: iffparse.library
Offset: -$DE
Syntax: StoreItemInContext(iff, localItem, contextNode)
C: VOID StoreItemInContext(struct IFFHandle *, struct LocalContextItem *, struct ContextNode *)
ML: StoreItemInContext(a0, a1, a2)
Arguments: iff = an IFFHandle structure for this context
localItem = the LocalContextItem to be stored
contextNode = the context node in which to store localItem
Result: none

StoreLocalItem (Revision 2.0)

Description: inserts a local context item into the context stack
Library: iffparse.library
Offset: -$D8
Syntax: error = StoreLocalItem(iff, localItem, position)
C: LONG StoreLocalItem(struct IFFHandle *, struct LocalContextItem *, LONG)
ML: d0 = StoreLocalItem(a0, a1, d0)
Arguments: iff = an IFFHandle structure
localItem = the LocalContextItem structure to insert
position = position to store the item
IFFSLI_ROOT ($00000001), IFFSLI_TOP ($00000002), IFFSLI_PROP ($00000003)
Result: error = zero if successful; nonzero error-code if unsuccessful

StrConvert (Revision 3.0)

Description: transforms a string according to collation information
Library: locale.library
Offset: -$AE
Syntax: length = StrConvert(locale, string, buffer, bufferSize, type)
C: ULONG StrConvert(struct Locale *, STRPTR, APTR, ULONG, ULONG)
ML: d0 = StrConvert(a0, a1, a2, d0, d1)
Arguments: locale = the locale to use for the transformation
string = the string to transform
buffer = buffer to receive the transformed string
bufferSize = size of buffer

type = the transformation to perform:
SC_ASCII ($00000000), SC_COLLATE1 ($00000001), SC_COLLATE2 ($00000002)

Result: length = length of the transformed string

StrlCmp (Revision 2.0)

Description: performs a case-insensitive string comparison
Library: utility.library
Offset: -$A2
Syntax: result = Stricmp(string1, string2)
C: LONG Stricmp(STRPTR, STRPTR)
ML: d0 = Stricmp(a0, a1)
Arguments: string1, string2 = the strings to compare
Result: result = negative if string1 < string2, zero if string1 = string2, positive if string1 > string2

StripFont (Revision 2.0)

Description: removes the tf_Extension from a font
Library: graphics.library
Offset: -$336
Syntax: StripFont(font)
C: VOID SyncSBitMap(struct Layer *)
ML: StripFont(a0)
Arguments: layer = locked Layer that has a SuperBitMap
Result: none

StrnCmp (Revision 3.0)

Description: performs a lexical string comparison based on a locale
Library: locale.library
Offset: -$B4
Syntax: result = StrnCmp(locale, string1, string2, length, type)
C: LONG StrnCmp(struct Locale *, STRPTR, STRPTR, LONG, ULONG)
ML: d0 = StrnCmp(a0, a1, a2, d0, d1)
Arguments: locale = the locale to use for this comparison
string1, string2 = the strings to compare
length = the maximum number of characters to compare;
-1 to compare all characters
type = the transformation to perform:
SC_ASCII ($00000000), SC_COLLATE1 ($00000001), SC_COLLATE2 ($00000002)
Result: result = negative if string1 < string2, zero if string1 = string2, positive if string1 > string2

Strnicmp (Revision 2.0)

Description: performs a length-limited, case-insensitive string comparison
Library: utility.library
Offset: -$A8
Syntax: result = Strnicmp(string1, string2, length)
C: LONG Strnicmp(STRPTR, STRPTR, LONG)
ML: d0 = Strnicmp(a0, a1, d0)
Arguments: string1, string2 = the strings to compare
length = maximum number of characters to examine
Result: result = negative if string1 < string2, zero if string1 = string2,
positive if string1 > string2

StrToDate (Revision 2.0)

Description: converts a string into a DateStamp
Library: dos.library
Offset: -$2EE
Syntax: success = StrToDate(datetime)
C: BOOL StrToDate(struct dateTime *)
ML: d0 = StrToDate(d1)
Arguments: dateTime = a DateTime structure with its dat_Format and dat_Flags fields
initialized and its dat_StrDate and dat_StrTime string fields set to
the desired date and time
Result: success = zero if unsuccessful

StrToLong (Revision 2.0)

Description: converts the decimal value of a string to a long (four-byte) value
Library: dos.library
Offset: -$330
Syntax: characters = StrToLong(string, value)
C: LONG StrToLong(STRPTR, LONG *)
ML: d0 = StrToLong(d1, d2)
Arguments: string = string representation of a decimal value
value = pointer to four bytes that will contain the converted value
Result: result = number of characters converted; -1 if unsuccessful (and value is set to 0)

SumKickData

Description: computes the checksum for the Kickstart delta list
Library: exec.library
Offset: -$264
Syntax: checksum = SumKickData( )
C: ULONG SumKickData(VOID)
ML: d0 = SumKickData( )
Arguments: none
Result: checksum = value to be stuffed into ExecBase->KickCheckSum

SumLibrary

Description: computes or checks the checksum on a library
Library: exec.library
Offset: -$1AA
Syntax: SumLibrary(library)
C: VOID SumLibrary(struct Library *)
ML: SumLibrary(a1)
Arguments: library = the library to be checked or changed
Result: none
If an old checksum is present and doesn't match the new one, Alert( ) will be called.

SuperState

Description: enters supervisor state with a user stack
Library: exec.library
Offset: -$96
Syntax: oldSysStack = SuperState( )
C: APTR SuperState(VOID)
ML: d0 = SuperState( )
Arguments: none
Result: oldSysStack = system stack pointer; zero if already in supervisor mode

Supervisor

Description: traps to a short supervisor mode function
Library: exec.library
Offset: -$1E
Syntax: result = Supervisor(userFunc)
C: ULONG Supervisor(VOID *)
ML: d0 = Supervisor(a5)
Arguments: userFunc = a short assembly language function ending in RTE
Result: result = whatever values the userFunc left in the registers

SwapBitsRastPortClipRect

Description: swaps bits between a common bitmap and an obscured ClipRect
Library: layers.library
Offset: -$7E
Syntax: SwapBitsRastPortClipRect(rp, cr)
C: VOID SwapBitsRastPortClipRect(struct RastPort *, struct ClipRect *)
ML: SwapBitsRastPortClipRect(a0, a1)
Arguments: rp = rastport containing data
cr = ClipRect with which to swap bits
Result: none

SyncSBitMap

Description: copies all bits from a layer's ClipRects into a SuperBitMap
Library: graphics.library

TagInArray (Revision 2.0)

Description: checks whether a tag value appears in an array of tag values
Library: utility.library
Offset: -$5A
Syntax: result = TagInArray(tagValue, tagArray)
C: BOOL TagInArray(Tag,Tag *)
ML: d0 = TagInArray(d0, a0)
Arguments: tagValue = tag value to search find
tagArray = array of tag values terminated by TAG_DONE
Result: result = nonzero if tagValue found

Text

Description: prints text using the current font
Library: graphics.library
Offset: -$3C
Syntax: Text(rastPort, string, length)
C: VOID Text(struct RastPort *, STRPTR, WORD)
ML: Text(a1, a0, d0)
Arguments: rastPort = RastPort to receive text
string = string to print
length = number of characters in string
Result: none

TextExtent (Revision 2.0)

Description: returns size information about a text string
Library: graphics.library
Offset: -$2B2
Syntax: TextExtent(rastPort, string, count, textExtent)
C: VOID textExtent(struct RastPort *, STRPTR, WORD, struct TextExtent *)
ML: TextExtent(a1, a0, d0, a2)
Arguments: rastPort = RastPort where text would be printed
string = string to check
count = number of characters in string
textExtent = TextExtent structure to receive the result
Result: none

TextFit (Revision 2.0)

Description: determines how many characters will fit within a given extent
Library: graphics.library
Offset: -$2B8
Syntax: chars = TextFit(rastport, string, count, textExtent, constrainingExtent, strDirection, constrainingBitWidth, constrainingBitHeight)
C: ULONG TextFit(struct RastPort *, STRPTR, UWORD, struct TextExtent *, struct TextExtent *, WORD, UWORD, UWORD)
ML: d0 = TextFit(d0, a1, a0, d0, a2, a3, d1, d2, d3)

Arguments: rastPort = RastPort where text would be printed
string = string to check
count = number of characters in string
textExtent = structure to receive the result
constrainingExtent = extent bounding the text
strDirection = offset to add to the string pointer to get to the next character in the string (usually 1)
constrainingBitWidth, constrainingBitHeight = alternative way to specify the box constraint that is independent of the rendering origin; 0-32767
Result: chars = number of characters from the origin of the given string that will fit in both the constraining extent and in the rendering width and height specified

TextLength

Description: returns the pixel width of a text string
Library: graphics.library
Offset: -$36
Syntax: length = TextLength(rastPort, string, count)
C: WORD TextLength(struct RastPort *, STRPTR, WORD)
ML: d0 = TextLength(a1, a0, d0)
Arguments: rastPort = RastPort where text would be printed
string = string to check
count = number of characters in string
Result: length = width of string in pixels

ThinLayerInfo (Obsolete under Revision 1.2)

Description: converts a 1.1 LayerInfo to a 1.0 LayerInfo.
Library: layers.library
Offset: -$A2
Syntax: ThinLayerInfo(li)
C: VOID ThinLayerInfo(struct Layer_Info *)
ML: ThinLayerInfo(a0)
Arguments: li = the LayerInfo structure to convert
Result: none

TimedDisplayAlert (Revision 3.0)

Description: displays a timed alert
Library: intuition.library
Offset: -$336
Syntax: response = TimedDisplayAlert(alertNumber, string, height, time)
C: BOOL TimedDisplayAlert(ULONG, UBYTE *, UWORD, ULONG)
ML: d0 = TimedDisplayAlert(d0, a0, d1, a1)

Arguments: alertNumber = bit flags specifying alert type:
RECOVERY_ALERT ($00000000), DEADEND_ALERT ($80000000)
string = message to be displayed by alert
height = minimum raster lines required to display alert message
time = length of time the alert should wait for user response; in jiffies
Result: response = nonzero if the left mouse button was pressed; zero if the right mouse button was pressed, if this is a DEADEND_ALERT, or the alert "timed out"

ToLower (Revision 2.0)
Description: converts a character to lower case
Library: utility.library
Offset: -$B4
Syntax: char = ToLower(char)
C: UBYTE ToLower(UBYTE)
ML: d0 = ToLower(d0)
Arguments: char = character to convert
Result: char = lower case version of char

ToUpper (Revision 2.0)
Description: converts a character to upper case
Library: utility.library
Offset: -$AE
Syntax: char = ToUpper(char)
C: UBYTE ToUpper(UBYTE)
ML: d0 = ToUpper(d0)
Arguments: char = character to convert
Result: char = upper case version of char

TypeOfMem
Description: determines the attributes of a given memory address
Library: exec.library
Offset: -$216
Syntax: attributes = TypeOfMem(address)
C: ULONG TypeOfMem(VOID *)
ML: d0 = TypeOfMem(a1)
Arguments: address = a memory address
Result: attributes = a longword of memory attribute flags; zero if the address is not in known RAM

UDivMod32 (Revision 2.0)
Description: performs an unsigned 32- by 32-bit division and modulus
Library: utility.library
Offset: -$9C
Syntax: quotient:remainder = UDivMod32(dividend, divisor)
C: ULONG:ULONG UDivMod32(ULONG, ULONG)

ML: d0/d1 = UDivMod32(d0, d1)
Arguments: dividend = unsigned 32-bit dividend
divisor = unsigned 32-bit divisor
Result: quotient = the unsigned 32-bit quotient of the division
remainder = the unsigned 32-bit remainder of the division

UMult32 (Revision 2.0)
Description: performs an unsigned 32- by 32-bit multiplication with 32-bit result
Library: utility.library
Offset: -$90
Syntax: result = UMult32(arg1, arg2)
C: ULONG UMult32(ULONG, ULONG)
ML: d0 = UMult32(d0, d1)
Arguments: arg1 = unsigned 32-bit multiplicand
arg2 = unsigned 32-bit multiplier
Result: result = the unsigned 32-bit product of arg1 and arg2

UMult64 (Revision 3.0)
Description: performs an unsigned 32- by 32-bit multiplication with 64-bit result
Library: utility.library
Offset: -$CC
Syntax: result = UMult64(arg1, arg2)
C: ULONG UMult64(ULONG, ULONG)
ML: d0/d1 = UMult64(d0, d1)
Arguments: arg1 = unsigned 32-bit multiplicand
arg2 = unsigned 32-bit multiplier
Result: result = the unsigned 64-bit product of arg1 and arg2

UnGetC (Revision 2.0)
Description: forces a character back into the input stream
Library: dos.library
Offset: -$13E
Syntax: value = UnGetC(fh, character)
C: LONG UnGetC(BPTR, LONG)
ML: d0 = UnGetC(d1, d2)
Arguments: fh = filehandle of file you're reading
character = character to put back into input stream or -1 to put last-read character back into input stream
Result: value = character pushed back into stream; zero if unsuccessful

UnLoadSeg
Description: unloads a seglist previously loaded by LoadSeg( )
Library: dos.library
Offset: -$9C
Syntax: success = UnLoadSeg(seglist)

C: BOOL UnLoadSeg(BPTR)
ML: d0 = UnLoadSeg(d1)
Arguments: seglist = seglist to unload
Result: success = zero if unsuccessful (return value is valid under Revision 2.0 only)

UnLock
Description: unlocks a file or directory
Library: dos.library
Offset: -$5A
Syntax: UnLock(lock)
C: VOID UnLock(BPTR)
ML: UnLock(d1)
Arguments: lock = lock to remove
Result: none

UnlockAmigaGuideBase (Revision 3.0)
Description: unlocks a client
Library: amigaguide.library
Offset: -$2A
Syntax: UnlockAmigaGuideBase(key)
C: VOID UnlockAmigaGuideBase(LONG)
ML: UnlockAmigaGuideBase(d0)
Arguments: key = value returned from LockAmigaGuideBase( )
Result: none

UnLockDosList (Revision 2.0)
Description: unlocks the DosList
Library: dos.library
Offset: -$294
Syntax: UnLockDosList(flags)
C: VOID UnLockDosList(ULONG)
ML: UnLockDosList(d1)
Arguments: flags = types of entries you want to unlock:
LDF_DEVICES ($00000004), LDF_VOLUMES ($00000008), LDF_ASSIGNS ($00000010), LDF_ENTRY ($00000020)
Result: none

UnlockIBase
Description: releases a lock on IntuitionBase
Library: intuition.library
Offset: -$1A4
Syntax: UnlockIBase(lock)
C: VOID UnlockIBase(ULONG)
ML: UnlockIBase(a0)

Arguments: lock = lock value returned by LockIBase( )
Result: none

UnlockLayer
Description: unlocks a layer
Library: layers.library
Offset: -$66
Syntax: UnlockLayer(layer)
C: VOID UnlockLayer(struct Layer *)
ML: UnlockLayer(a0)
Arguments: layer = the layer to unlock
Result: none

UnlockLayerInfo
Description: unlocks a LayerInfo structure
Library: layers.library
Offset: -$8A
Syntax: UnlockLayerInfo(li)
C: VOID UnlockLayerInfo(struct Layer_Info *)
ML: UnlockLayerInfo(a0)
Arguments: li = the Layer_Info structure to unlock
Result: none

UnlockLayerRom
Description: unlocks a Layer structure
Library: graphics.library
Offset: -$1B6
Syntax: UnlockLayerRom(layer)
C: VOID UnlockLayerRom(struct Layer *)
ML: UnlockLayerRom(a5)
Arguments: layer = Layer structure previously locked with the LockLayerROM( ) function
Result: none

UnlockLayers
Description: unlocks all layers
Library: layers.library
Offset: -$72
Syntax: UnlockLayers(li)
C: VOID UnlockLayers(struct Layer_Info *)
ML: UnlockLayers(a0)
Arguments: li = the Layer_Info structure containing the list of layers to unlock
Result: none

UnlockPubScreen (Revision 2.0)

Description: releases a lock on a public screen
Library: intuition.library
Offset: -$204
Syntax: UnlockPubScreen(name, screen)
C: VOID UnlockPubScreen(UBYTE *, struct Screen *)
ML: UnlockPubScreen(a0, a1)
Arguments: name = name of a public screen; may be 0
screen = public screen pointer as returned by LockPubScreen( );
used only if name is 0
Result: none

UnlockPubScreenList (Revision 2.0)

Description: releases a public screen list
Library: intuition.library
Offset: -$210
Syntax: UnlockPubScreenList( )
C: VOID UnlockPubScreenList(VOID)
ML: UnlockPubScreenList( )
Arguments: none
Result: none

UnLockRecord (Revision 2.0)

Description: unlocks a record
Library: dos.library
Offset: -$11A
Syntax: success = UnLockRecord(fh, offset, length)
C: BOOL UnLockRecord(BPTR, ULONG, ULONG)
ML: d0 = UnLockRecord(d1, d2, d3)
Arguments: fh = filehandle of locked file
offset = record's starting position in file
length = size of record
Result: success = zero if unsuccessful

UnLockRecords (Revision 2.0)

Description: unlocks a list of records
Library: dos.library
Offset: -$120
Syntax: success = UnLockRecords(recordArray)
C: BOOL UnLockRecords(struct RecordLock *)
ML: d0 = UnLockRecords(d1)
Arguments: recordArray = list of records to be unlocked
Result: success = zero if unsuccessful

UnlockRexxBase (Revision 2.0)

Description: releases a semaphore lock on the RexxBase structure
Library: rexxsyslib.library
Offset: -$1C8
Syntax: UnlockRexxBase(resource)
C: VOID UnlockRexxBase(ULONG)
ML: UnlockRexxBase(d0)
Arguments: resource = which resources to unlock; must be 0 (all resources)
Result: none

UnpackStructureTags (Revision 3.0)

Description: unpacks a structure to values in a tag list
Library: utility.library
Offset: -$D8
Syntax: num = UnpackStructureTags(pack, packtable, tags)
C: ULONG UnpackStructureTags(APTR, ULONG *, struct TagItem *)
ML: d0 = UnpackStructureTags(a0, a1, a2)
Arguments: pack = the data area to unpack
packtable = the packing information table:
see <utility/pack.h> for description
tags = the tag list to initialize
Result: num = number of tag items unpacked

UpfrontLayer

Description: moves a layer in front of all other layers
Library: layers.library
Offset: -$30
Syntax: success = UpfrontLayer(dummy, layer)
C: LONG UpfrontLayer(LONG, struct Layer *)
ML: d0 = UpfrontLayer(a0, a1)
Arguments: dummy = unused
layer = the layer to move; must not be a backdrop layer
Result: success = zero if unsuccessful

UserState

Description: returns to user state with a user stack
Library: exec.library
Offset: -$9C
Syntax: UserState(sysStack)
C: VOID UserState(APTR)
ML: UserState(d0)
Arguments: sysStack = supervisor stack pointer
Result: none

Vacate (Revision 3.0)

Description: releases a bidMessage from Procure( )
Library: exec.library
Offset: -$222
Syntax: Vacate(semaphore, bidMessage)
C: VOID Vacate(struct SignalSemaphore *, struct SemaphoreMessage *)
ML: Vacate(a0, a1)
Arguments: semaphore = the SignalSemaphore to Vacate( )
bidMessage= the SemaphoreMessage to abort
Result: none

VBeamPos

Description: returns the raster beam's current vertical position
Library: graphics.library
Offset: -$180
Syntax: pos = VBeamPos( )
C: LONG VBeamPos(VOID)
ML: d0 = VBeamPos( )
Arguments: none
Result: pos = vertical position of raster beam

VFPrintf (Revision 2.0)

Description: formats and prints a string to a file
Library: dos.library
Offset: -$162
Syntax: count = VFPrintf(fh, fmt, argv)
C: LONG VFPrintf(BPTR, STRPTR, ...)
ML: d0 = VFPrintf(d1, d2, d3)
Arguments: fh = filehandle to receive string
fmt = RawDoFmt( )-style formatting string
argv = array of values to format and print
Result: count = number of bytes written; -1 if unsuccessful

VFWritef (Revision 2.0)

Description: writes a BCPL-formatted string to a file
Library: dos.library
Offset: -$15C
Syntax: count = VFWritef(fh, fmt, argv)
C: LONG VFWritef(BPTR, STRPTR, ...)
ML: d0 = VFWritef(d1, d2, d3)
Arguments: fh = filehandle to write to
fmt = BCPL-style formatting string
argv = array of values to write
Result: count = number of bytes written; -1 if unsuccessful

VideoControl (Revision 2.0)

Description: modifies the operation of a ViewPort's ColorMap
Library: graphics.library
Offset: -$2C4
Syntax: error = VideoControl(colorMap , tags)
C: ULONG VideoControl(struct ColorMap *, struct TagItem *)
ML: d0 = VideoControl(a0, a1)
Arguments: colorMap = ColorMap structure
tags = tag list of attributes—see <graphics/videocontrol.h> for description
Result: error = zero if successful

ViewAddress

Description: returns Intuition's View structure
Library: intuition.library
Offset: -$126
Syntax: view = ViewAddress( )
C: struct View *ViewAddress(VOID)
ML: d0 = ViewAddress( )
Arguments: none
Result: view = the Intuition View structure

ViewPortAddress

Description: returns a window's ViewPort structure
Library: intuition.library
Offset: -$12C
Syntax: viewPort = ViewPortAddress(window)
C: struct ViewPort *ViewPortAddress(struct Window *)
ML: d0 = ViewPortAddress(a0)
Arguments: window = window to interrogate
Result: viewPort = the window's ViewPort structure

VPrintf (Revision 2.0)

Description: formats and prints a string to the default output device
Library: dos.library
Offset: -$3BA
Syntax: count = VPrintf(fmt, argv)
C: LONG VPrintf(STRPTR, ...)
ML: d0 = VPrintf(d1, d2)
Arguments: fmt = RawDoFmt( )-style formatting string
argv = array of values to format and print
Result: count = number of bytes written; -1 if unsuccessful

Wait

Description: waits for one or more signals
Library: exec.library

Offset: -$13E
Syntax: signals = Wait(signalSet)
C: ULONG Wait(ULONG)
ML: d0 = Wait(d0)
Arguments: signalSet = the set of signals for which to wait
Result: signals = the set of signals that were active

WaitBlit
Description: waits for the blitter to finish its current task
Library: graphics.library
Offset: -$E4
Syntax: WaitBlit( )
C: VOID WaitBlit(VOID)
ML: WaitBlit( )
Arguments: none
Result: none

WaitBOVP
Description: waits until the raster beam reaches bottom of the viewport
Library: graphics.library
Offset: -$192
Syntax: WaitBOVP(viewPort)
C: VOID WaitBOVP(struct ViewPort *)
ML: WaitBOVP(a0)
Arguments: viewPort = ViewPort structure
Result: none

WaitForChar
Description: waits a specified amount of time for a character to become available from a file or device
Library: dos.library
Offset: -$CC
Syntax: status = WaitForChar(fh, timeout)
C: BOOL WaitForChar(BPTR, LONG)
ML: d0 = WaitForChar(d1, d2)
Arguments: fh = filehandle of input file
timeout = number of microseconds to wait
Result: status = nonzero if a character did indeed arrive in time

WaitIO
Description: waits for completion of an I/O request
Library: exec.library
Offset: -$1DA
Syntax: error = WaitIO(iORequest)
C: BYTE WaitIO(struct IORequest *)

ML: d0 = WaitIO(a1)
Arguments: iORequest = pointer to an IORequest block
Result: error = zero if successful; a sign extended copy of io_Error if unsuccessfull

WaitPkt (Revision 2.0)
Description: waits for a packet to arrive at your pr_MsgPort
Library: dos.library
Offset: -$FC
Syntax: packet = WaitPkt( )
C: struct DosPacket *WaitPkt(VOID)
ML: d0 = WaitPkt( )
Arguments: none
Result: packet = the much-anticipated packet

WaitPort
Description: waits for a given port to be non-empty
Library: exec.library
Offset: -$180
Syntax: message = WaitPort(port)
C: struct Message *WaitPort(struct MsgPort *)
ML: d0 = WaitPort(a0)
Arguments: port = the message port to wait for
Result: message = the first available message

WaitTOF
Description: waits until the raster beam reaches the top of the viewport
Library: graphics.library
Offset: -$10E
Syntax: WaitTOF( )
C: VOID WaitTOF(VOID)
ML: WaitTOF( )
Arguments: none
Result: none

WBenchToBack
Description: moves the Workbench screen behind all other screens
Library: intuition.library
Offset: -$150
Syntax: success = WBenchToBack( )
C: BOOL WBenchToBack(VOID)
ML: d0 = WBenchToBack( )
Arguments: none
Result: success = zero if unsuccessful

WBInfo (Revision 3.0)

Description: brings up the Workbench information requester
Library: workbench.library
Offset: -$5A
Syntax: success = WBInfo(lock, name, screen)
C: ULONG WBInfo(BPTR, STRPTR, struct Screen *)
ML: d0 = WBInfo(a0, a1, a2)
Arguments: lock = lock on the directory containing the icon
name = name of the icon about which to get information
screen = screen on which the requester is to be displayed
Result: success = zero if unsuccessful

WhichLayer

Description: returns the layer containing a particular point
Library: layers.library
Offset: -$84
Syntax: layer = WhichLayer(li, x, y)
C: struct Layer *WhichLayer(struct Layer_Info *, WORD, WORD)
ML: d0 = WhichLayer(a0, d0, d1)
Arguments: li = the LayerInfo structure to check
x, y = coordinates in the BitMap of the point to check
Result: layer = the topmost layer containing this point; zero if unsuccessful

WindowToBack

Description: moves a window behind all other screens
Library: intuition.library
Offset: -$132
Syntax: WindowToBack(window)
C: VOID WindowToBack(struct Window *)
ML: WindowToBack(a0)
Arguments: window = window to be pushed to back
Result: none

WindowToFront

Description: moves a window in front of all other screens
Library: intuition.library
Offset: -$138
Syntax: WindowToFront(window)
C: VOID WindowToFront(struct Window *)
ML: WindowToFront(a0)
Arguments: window = window to be be popped to the front
Result: none

Write

Description: writes bytes of data to a file
Library: dos.library
Offset: -$30
Syntax: actualLength = Write(fh, buffer, length)
C: LONG Write(BPTR, VOID *, LONG)
ML: d0 = Write(d1, d2, d3)
Arguments: fh = filehandle of file to write to
buffer = buffer containing data to write
length = number of bytes to write
Result: actualLength = number of bytes successfully written; -1 if unsuccessful

WriteChars (Revision 2.0)

Description: writes bytes to the default output device
Library: dos.library
Offset: -$3AE
Syntax: returnedLength = WriteChars(buffer, length)
C: LONG WriteChars(STRPTR, LONG)
ML: d0 = WriteChars(d0, d1)
Arguments: fh = filehandle of file to receive data
buffer = buffer containing data to write
length = number of bytes to write
Result: returnedLength = number of bytes successfully written;
-1 if an error occured

WriteChunkBytes (Revision 2.0)

Description: writes data from a buffer into the current chunk.
Library: iffparse.library
Offset: -$42
Syntax: error = WriteChunkBytes(iff, buf, numBytes)
C: LONG WriteChunkBytes(struct IFFHandle *, APTR, LONG)
ML: d0 = WriteChunkBytes(a0, a1, d0)
Arguments: iff = an IFFHandle structure
buf = memory to write
numBytes = number of bytes to write
Result: error = actual number of bytes written, if successful; negative if an error occurs

WriteChunkRecords (Revision 2.0)

Description: writes records from a buffer to the current chunk
Library: iffparse.library
Offset: -$4E
Syntax: error = WriteChunkRecords(iff, buf, recsize, numrec)
C: LONG WriteChunkRecords(struct IFFHandle *, APTR, LONG, LONG)
ML: d0 = WriteChunkRecords(a0, a1, d0, d1)

Arguments: iff = an IFFHandle structure
buf = memory to write
recsize = size of data records to write
numrec = number of data records to write
Result: error = actual number of whole records written, if successful; negative if an error occurs

WriteExpansionByte

Description: writes a byte to expansion memory
Library: expansion.library
Offset: -$72
Syntax: WriteExpansionByte(board, offset, byte)
C: VOID WriteExpansionByte(APTR, ULONG, ULONG)
ML: WriteExpansionByte(a0, d0, d1)
Arguments: board = base pointer to a new style expansion board
offset = logical offset from the configdev base
byte = the byte of data to be written
Result: none

WritePixel

Description: sets a pixel to the current pen color
Library: graphics.library
Offset: -$144
Syntax: error = WritePixel(rastPort, x, y)
C: LONG WritePixel(struct RastPort *, WORD, WORD)
ML: d0 = WritePixel(a1, d0, d1)
Arguments: rastPort = RastPort structure
x, y = point to set
Result: error = zero if successfull

WritePixelArray8

Description: draws the pixels found in a rectangular array
Library: graphics.library
Offset: -$312
Syntax: count = WritePixelArray8(rastPort, x1, y1, x2, y2, array, tempRastPort)
C: LONG WritePixelArray8(struct RastPort *, UWORD, UWORD, UWORD, UWORD, UBYTE *, struct RastPort *)
ML: d0 = WritePixelArray8(a0, d0, d1, d2, d3, a2, a1)
Arguments: rastPort = RastPort structure
x1, y1 = top-left point in the RastPort
x2, y2 = bottom-right point in the RastPort
array = array to hold pixel color values
tempRastPort = temporary RastPort structure
Result: count = number of pixels drawn

WritePixelLine8

Description: draws the pixels found in an array
Library: graphics.library
Offset: -$306
Syntax: count = WritePixelLine8(rastPort, x1, y1, width, array, temprastPort)
C: LONG WritePixelLine8(struct RastPort *, UWORD, UWORD, UWORD, UBYTE *, struct RastPort *)
ML: d0 = WritePixelLine8(a0, d0, d1, d2, a2, a1)
Arguments: rastPort = RastPort structure
x, y = starting point of line
width = number of horizontal pixels to read
array = array to hold pixel color values
temprastPort = temporary RastPort structure
Result: count = number of pixels drawn

XorRectRegion

Description: XORs a rectangle with a region, leaving the result in the region
Library: graphics.library
Offset: -$22E
Syntax: success = XorRectRegion(region, rectangle)
C: BOOL XorRectRegion(struct Region *, struct Rectangle *)
ML: d0 = XorRectRegion(a0, a1)
Arguments: region = Region structure
rectangle = Rectangle structure
Result: success = zero if unsuccessful

XorRegionRegion

Description: XORs two regions together, leaving the result in the second region
Library: graphics.library
Offset: -$26A
Syntax: success = XorRegionRegion(region1, region2)
C: BOOL XorRegionRegion(struct Region *, struct Region *)
ML: d0 = XorRegionRegion(a0, a1)
Arguments: region1 = Region structure
region2 = destination Region structure
Result: success = zero if unsuccessful

ZipWindow (Revision 2.0)

Description: restores a window's previous size and position
Library: intuition.library
Offset: -$1F8
Syntax: ZipWindow(window)
C: VOID ZipWindow(struct Window *)
ML: ZipWindow(a0)

Arguments: window = window to "zip"
Result: none
The window's size and position are restored to their values at the time of the last call to ZipWindow( ).

2
Structures

If you want to program the Amiga, you'll need to learn all about its structures. On the Amiga, a structure is a logical grouping of variables. For example, there is a structure called "Screen" for each of the Amiga's movable, draggable system screens. Through the Screen structure, you can find out the size of the screen, the font being used for the screen, and many other useful pieces of information. You can also change many of these parameters, either by writing values into the Screen structure, or by calling an Amiga function which alters the values.

Amiga structures are defined in the include files that Commodore provides. Each new version of the operating system brings new include files. The includes are available from Commodore. You may also get them when you buy your assembler or compiler. Commodore maintains two sets of include files: one for the C language and one for assembly language (also known as ML or machine language).

To learn about structured data types in C, consult the structures section of a C programming book. Machine language has no built-in structured data types, so macros are used to build structures. The STRUCTURE and STRUCT macros are used. STRUCTURE is used to start the definition, and STRUCTs are used to represent structures within other structures. Here's an example:

```
STRUCTURE DBP,0
    WORD  dbp_BufY
    WORD  dbp_BufX
    APTR  dbp_BufPath
    APTR  dbp_BufBuffer
    APTR  dbp_BufPlanes
    LABEL dbp_SIZEOF
```

The zero after the DBP is a offset used to get to the first variable inside the structure (dbp_BufY, in this case). This value is usually zero. Structures end with a LABEL directive, which is used to define the relative address from the start of the structure. Thus, it returns the size of the structure when it's placed at the definition's end.

Here is a sample structure, Resident, which is used to make programs that can survive a reboot.

Resident

(RT in ML)
26 bytes

C Include File: exec/resident.h
ML Include File: exec/resident.i

byte	hex	C name	C type	ML name	ML type
0	00	rt_MatchWord	UWORD	RT_MATCHWORD	UWORD
2	02	rt_MatchTag	struct Resident *	RT_MATCHTAG	APTR
6	06	rt_EndSkip	APTR	RT_ENDSKIP	APTR
10	0A	rt_Flags	UBYTE	RT_FLAGS	UBYTE
11	0B	rt_Version	UBYTE	RT_VERSION	UBYTE
12	0C	rt_Type	UBYTE	RT_TYPE	UBYTE
13	0D	rt_Pri	BYTE	RT_PRI	BYTE
14	0E	rt_Name	char *	RT_NAME	APTR
18	12	rt_IdString	char *	RT_IDSTRING	APTR
22	16	rt_Init	APTR	RT_INIT	APTR

rt_Flags

C name	ML name	bit	decimal	hex
RTF_COLDSTART	RT,COLDSTART	0	1	01
RTF_AUTOINIT	RT,AUTOINIT	7	128	80

First comes the name of the structure. In this case, the C and ML versions have different names.

Second comes the size of the structure, 26 bytes. Note that structure sizes can change when a new version of the operating system comes out.

If the structure has changed as the operating system has matured, the nature of the change will come next. For instance, a structure may be "new to 2.0," or "new to 3.0," or updated for 2.0 or 3.0. This book reflects the latest operating system, 3.0. Often the changes are minor (a definition was declared as a SHORT, but now it's declared as a WORD). Often, new fields are added to a structure, making it longer.

Next comes the name of the include file where the definition can be found. The C and ML names usually differ only in the final character of the filename (h for C includes and i for machine language includes).

Next comes the variable definitions. Both decimal and hexadecimal (abbreviated

as "hex") offsets are given. The names and type of the C and ML fields are given side-by-side. It's often illuminating to look at both the definitions—often the C definition is more descriptive and the ML definition is more concrete.

Notes come next. These usually indicate a naming oddity. Sometimes a field has two names and it's mentioned here in the note.

Finally, if any flags with symbolic defintions appear in the structure, they are listed with their bit, decimal, and hex values. Note that some flags are defined as bit flags (they refer to bit numbers) and others are values. If they are bit flags, they describe which single bit to set or check.

AChain

274 bytes
(Update for 3.0)

C Include File: dos/dosasl.h
ML Include File: dos/dosasl.i

byte	hex	C name	C type	ML type
0	0000	an_Child	struct AChain *	CPTR
4	0004	an_Parent	struct AChain *	CPTR
8	0008	an_Lock	BPTR	LONG
12	000C	an_Info	struct FileInfoBlock	STRUCT
272	0110	an_Flags	BYTE	BYTE
273	0111	an_String	UBYTE[1]	LABEL

an_Flags

C flag name	C bit name	ML bit name	bit	decimal	hex
DDF_PatternBit	DDB_PatternBit	DD,PatternBit	0	1	01
DDF_ExaminedBit	DDB_ExaminedBit	DD,ExaminedBit	1	2	02
DDF_Completed	DDB_Completed	DD,Completed	2	4	04
DDF_AllBit	DDB_AllBit	DD,AllBit	3	8	08
DDF_Single	DDB_Single	DD,SINGLE	4	16	10

AmigaGuideHost

40 bytes
(New to 3.0)

C Include File: libraries/amigaguide.h
ML Include File: libraries/amigaguide.i

byte	hex	C name	C type	ML type
0	00	agh_Dispatcher	struct Hook	STRUCT
20	14	agh_Reserved	ULONG	ULONG
24	18	agh_Flags	ULONG	ULONG
28	1C	agh_UseCnt	ULONG	ULONG
32	20	agh_SystemData	APTR	APTR
36	24	sgh_UserData	APTR	APTR

AmigaGuideMsg

52 bytes
(New to 3.0)

C Include File: libraries/amigaguide.h
ML Include File: libraries/amigaguide.i

byte	hex	name	C type	ML type
0	00	agm_Msg	struct Message	STRUCT
20	14	agm_Type	ULONG	ULONG
24	18	agm_Data	APTR	APTR
28	1C	agm_DSize	ULONG	ULONG
32	20	agm_DType	ULONG	ULONG
36	24	agm_Pri_Ret	ULONG	ULONG
40	28	agm_Sec_Ret	ULONG	ULONG
44	2C	agm_System1	APTR	APTR
48	30	agm_System2	APTR	APTR

AnalogSignalInterval

4 bytes
(New to 2.0)

C Include File: graphics/monitor.h
ML Include File: graphics/monitor.i

byte	hex	name	type
0	00	asi_Start	UWORD
2	02	asi_Stop	UWORD

AnchorPath

282 bytes
(Updated for 3.0)

C Include File: dos/dosasl.h
ML Include File: dos/dosasl.i

byte	hex	name	C type	ML type
0	0000	ap_Base	struct AChain *	CPTR
4	0004	ap_Last	struct AChain *	CPTR
8	0008	ap_BreakBits	LONG	LONG
12	000C	ap_FoundBreak	LONG	LONG
16	0010	ap_Flags	byte	BYTE
17	0011	ap_Reserved	byte	BYTE
18	0012	ap_Strlen	WORD	WORD
20	0014	ap_Info	struct FileInfoBlock	STRUCT
280	0118	ap_Buf	UBYTE[1]	LABEL

Note: ap_Base and ap_Last are also known as ap_First and ap_Current, respectively.

ap_Flags

C flag name	C bit name	ML bit name	bit	decimal	hex
APF_DOWILD	APB_DOWILD	AP,DOWILD	0	1	01
APF_ITSWILD	APB_ITSWILD	AP,ITSWILD	1	2	02
APF_DODIR	APB_DODIR	AP,DODIR	2	4	04
APF_DIDDIR	APB_DIDDIR	AP,DIDDIR	3	8	08
APF_NOMEMERR	APB_NOMEMERR	AP,NOMEMERR	4	16	10
APF_DODOT	APB_DODOT	AP,DODOT	5	32	20
APF_DirChanged	APB_DirChanged	AP,DirChanged	6	64	40
APF_FollowHLinks	APB_FollowHLinks	AP,FollowHLinks	7	128	80

AC

see AnimComp

AF

see AvailFonts

AFH

see AvailFontsHeader

AnimComp

(AC in ML)
38 bytes

C Include File: graphics/gels.h
ML Include File: graphics/gels.i

byte	hex	C name	C type	ML name	ML type
0	00	Flags	WORD	ac_CompFlags	WORD
2	02	Timer	WORD	ac_Timer	WORD
4	04	TimeSet	WORD	ac_TimeSet	WORD
6	06	NextComp	struct AnimComp *	ac_NextComp	APTR
10	0A	PrevComp	struct AnimComp *	ac_PrevComp	APTR
14	0E	NextSeq	struct AnimComp *	ac_NextSeq	APTR
18	12	PrevSeq	struct AnimComp *	ac_PrevSeq	APTR
22	16	AnimCRoutine	WORD(*)()	ac_AnimCRoutine	APTR
26	1A	YTrans	WORD	ac_YTrans	WORD
28	1C	XTrans	WORD	ac_XTrans	WORD
30	1E	HeadOb	struct AnimOb *	ac_HeadOb	APTR
34	22	AnimBob	struct Bob *	ac_AnimBob	APTR

Flags (ac_CompFlags in ML)

name	bit	decimal	hex
RINGTRIGGER	0	1	01

AnimOb

(AO in ML)
42 bytes

C Include File: graphics/gels.h
ML Include File: graphics/gels.i

byte	hex	C name	C type	ML name	ML type
0	00	NextOb	struct AnimOb *	ao_NextOb	APTR
4	04	PrevOb	struct AnimOb *	ao_PrevOb	APTR
8	08	Clock	LONG	ao_Clock	LONG
12	0C	AnOldY	WORD	ao_AnOldY	WORD
14	0E	AnOldX	WORD	ao_AnOldX	WORD
16	10	AnY	WORD	ao_AnY	WORD
18	12	AnX	WORD	ao_AnX	WORD
20	14	YVel	WORD	ao_YVel	WORD
22	16	XVel	WORD	ao_XVel	WORD

byte	hex	C name	C type	ML name	ML type
24	18	XAccel	WORD	ao_XAccel	WORD
26	1A	YAccel	WORD	ao_YAccel	WORD
28	1C	RingYTrans	WORD	ao_RingYTrans	WORD
30	1E	RingXTrans	WORD	ao_RingXTrans	WORD
32	20	AnimORoutine	WORD (*)()	ao_AnimORoutine	APTR
36	24	HeadComp	struct AnimComp *	ao_HeadComp	APTR
40	28	AUserExt	AUserStuff	ao_AUserExt	LABEL

AO

see AnimOb

AppIcon

32 bytes
(New to 2.0)

C Include File: workbench/workbench.h
ML Include File: workbench/workbench.i

byte	hex	name
0	00	ai_Node
14	0E	ai_Ptr
18	12	ai_MsgPort
22	16	ai_UserData
26	1a	ai_ID
30	1e	ai_Version

Note: This structure is private to the operating system. It is not to be used by application programmers. This is the 2.0 version. In 3.0, the fields are unlisted.

AppMenuItem

32 bytes
(New to 2.0)

C Include File: workbench/workbench.h
ML Include File: workbench/workbench.i

byte	hex	name
0	00	ami_Node
14	0E	ami_Ptr
18	12	ami_MsgPort
22	16	ami_UserData
26	1a	ami_ID
30	1e	ami_Version

Note: This structure is private to the operating system. It is not to be used by application programmers. This is the 2.0 version. In 3.0, the fields are unlisted.

AppMessage

86 bytes
(New to 2.0)

C Include File: workbench/workbench.h
ML Include File: workbench/workbench.i

byte	hex	name	C type	ML type
0	00	am_Message	struct Message	STRUCT
20	14	am_Type	UWORD	UWORD
22	16	am_UserData	ULONG	ULONG
26	1A	am_ID	ULONG	ULONG
30	1E	am_NumArgs	LONG	LONG
34	22	am_ArgList	struct WBArg *	APTR
38	26	am_Version	UWORD	UWORD
40	28	am_Class	UWORD	UWORD
42	2A	am_MouseX	WORD	WORD
44	2C	am_MouseY	WORD	WORD
46	2E	am_Seconds	ULONG	ULONG
50	32	am_Micros	ULONG	ULONG
54	36	am_Reserved	ULONG[8]	STRUCT

AppWindow

32 bytes
(New to 2.0)

C Include File: workbench/workbench.h
ML Include File: workbench/workbench.i

byte	hex	name
0	00	ami_Node
14	0E	ami_Ptr
18	12	ami_MsgPort
22	16	ami_UserData
26	1a	ami_ID
30	1e	ami_Version

Note: This structure is private to the operating system. It is not to be used by application programmers. This is the 2.0 version. In 3.0, the fields are unlisted.

AreaInfo

24 bytes
(updated for 3.0)

C Include File: graphics/rastport.h
ML Include File: graphics/rastport.i

byte	hex	C name	C type	ML name	ML type
0	00	VctrTbl	WORD *	ai_VctrTbl	LONG
4	04	VctrPtr	WORD *	ai_VctrPtr	LONG
8	08	FlagTbl	BYTE *	ai_FlagTbl	LONG
12	0C	FlagPtr	BYTE *	ai_FlagPtr	LONG
16	10	Count	WORD	ai_Count	WORD
18	12	MaxCount	WORD	ai_MaxCount	WORD
20	14	FirstX	WORD	ai_FirstX	WORD
22	16	FirstY	WORD	ai_FirstY	WORD

AudChannel

16 bytes

C Inclue File: hardware/custom.h
ML Include File: hardware/custom.i

byte	hex	name	ML type	ML type
0	00	ac_ptr	UWORD *	$00
4	04	ac_len	UWORD	$04
6	06	ac_per	UWORD	$06
8	08	ac_vol	UWORD	$08
10	0A	ac_dat	UWORD	$0A
12	0C	ac_pad	UWORD[2]	

AvailFonts

(AF in ML)
10 bytes
(Updated for 3.0)

C Include File: diskfont/diskfont.h
ML Include File: diskfont/diskfont.i

byte	hex	name	C type	ML type
0	00	af_Type	UWORD	UWORD
2	02	af_Attr	struct TextAttr	STRUCT

af_Type

C flag name	C bit name	ML bit name	bit	decimal	hex
AFF_MEMORY	AFB_MEMORY	AF,MEMORY	0	1	000001
AFF_DISK	AFB_DISK	AF,DISK	1	2	000002
AFF_SCALED	AFB_SCALED	AF,SCALED	2	4	000004
AFF_TAGGED	AFB_TAGGED	AF,TTATTR	16	65536	010000

Note: This structure used to be located in libraries/diskfont.h (C version) and libraries/diskfont.i (ML version).

AvailFontsHeader

(AFH in ML)
2 bytes
(Updated for 3.0)

C Include File: diskfont/diskfont.h
ML Include File: diskfont/diskfont.i

byte	hex	name	C type	ML type
0	00	afh_NumEntries	UWORD	UWORD

Note: This structure used to be located in libraries/diskfont.h (C version) and libraries/diskfont.i (ML version).

BadBlockBlock

512 bytes
(New to 2.0)

C Include File: devices/hardblocks.h
ML Include File: devices/hardblocks.i

byte	hex	name	C type	ML type
0	00	bbb_ID	ULONG	ULONG
4	04	bbb_SummedLongs	ULONG	ULONG
8	08	bbb_ChkSum	LONG	LONG
12	0C	bbb_HostID	ULONG	ULONG
16	10	bbb_Next	ULONG	ULONG
20	14	bbb_Reserved	ULONG	ULONG
24	18	bbb_BlockPairs	struct BadBlockEntry	STRUCT

Note: ID_BADBLOCK is defined as 0x42414442 ("BADB").

BadBlockEntry

8 bytes
(New to 2.0)

C Include File: devices/hardblocks.h
ML Include File: devices/hardblocks.i

byte	hex	name	type
0	00	bbe_BadBlock	ULONG
4	04	bbe_GoodBlock	ULONG

BB

see BootBlock

BitMap

40 bytes

C Include File: graphics/gfx.h
ML Include File: graphics/gfx.i

byte	hex	C name	C type	ML name	ML type
0	00	BytesPerRow	UWORD	bm_BytesPerRow	WORD
2	02	Rows	UWORD	bm_Rows	WORD
4	04	Flags	UBYTE	bm_Flags	BYTE
5	05	Depth	UBYTE	bm_Depth	BYTE
6	06	pad	UWORD	bm_pad	WORD
8	08	Planes	PLANEPTR[8]	bm_Planes	STRUCT

BitMapHeader

20 bytes
(New to 3.0)

C Include File: datatypes/pictureclass.h
ML Include File: datatypes/pictureclass.i

byte	hex	name	type
0	00	bmh_Width	UWORD
2	02	bmh_Height	UWORD
4	04	bmh_Left	WORD
6	06	bmh_Top	WORD
8	08	bmh_Depth	UBYTE
9	09	bmh_Masking	UBYTE
10	0A	bmh_Compression	UBYTE
11	0B	bmh_Pad	UBYTE
12	0C	bmh_Transparent	UWORD
14	0E	bmh_XAspect	UBYTE
15	0F	bmh_YAspect	UBYTE
16	10	bmh_PageWidth	WORD
18	12	bmh_PageHeight	WORD

BitScaleArgs

48 bytes
(New to 2.0)

C Include File: graphics/scale.h
ML Include File: graphics/scale.i

byte	hex	name	C type	ML type
0	00	bsa_SrcX	UWORD	UWORD
2	02	bsa_SrcY	UWORD	UWORD
4	04	bsa_SrcWidth	UWORD	UWORD
6	06	bsa_SrcHeight	UWORD	UWORD
8	08	bsa_XSrcFactor	UWORD	UWORD
10	0A	bsa_YSrcFactor	UWORD	UWORD
12	0C	bsa_DestX	UWORD	UWORD
14	0E	bsa_DestY	UWORD	UWORD
16	10	bsa_DestWidth	UWORD	UWORD
18	12	bsa_DestHeight	UWORD	UWORD
20	14	bsa_XDestFactor	UWORD	UWORD
22	16	bsa_YDestFactor	UWORD	UWORD
24	18	bsa_SrcBitMap	struct BitMap *	APTR
28	1C	bsa_DestBitMap	struct BitMap *	APTR
32	20	bsa_Flags	ULONG	ULONG
36	24	bsa_XDDA	UWORD	UWORD
38	26	bsa_YDDA	UWORD	UWORD
40	28	bsa_Reserved1	LONG	LONG
44	2C	bsa_Reserved2	LONG	LONG

Note: As of version 3.0, bsa_Flags is reserved and must be set to 0.

bltnode

18 bytes

C Include File: hardware/blit.h
ML Include File: hardware/blit.i

byte	hex	C name	C type	ML name	ML type
0	00	n	struct bltnode *	bn_n	LONG
4	04	function	int(*)()	bn_function	LONG
8	08	stat	char	bn_stat	BYTE
9	09			bn_dummy	BYTE
10	0A	blitsize	short	bn_blitsize	WORD
12	0C	beamsync	short	bn_beamsync	WORD
14	0E	cleanup	int(*)()	bn_cleanup	LONG

stat (bn_stat in ML)

C flag name	ML bit name	ML flag name	bit	decimal	hex
CLEANME	CLEANME	CLEANMEn	6	64	40

Bob

32 bytes

C Include File: graphics/gels.h
ML Include File: graphics/gels.i

byte	hex	C name	C type	ML name	ML type
0	00	Flags	WORD	bob_BobFlags	WORD
2	02	SaveBuffer	WORD *	bob_SaveBuffer	APTR
6	06	ImageShadow	WORD *	bob_ImageShadow	APTR
10	0A	Before	struct Bob *	bob_Before	APTR
14	0E	After	struct Bob *	bob_After	APTR
18	12	BobVSprite	struct VSprite *	bob_BobVSprite	APTR
22	16	BobComp	struct AnimComp *	bob_BobComp	APTR
26	1A	DBuffer	struct DBufPacket *	bob_DBuffer	APTR
30	1E	BUserExt	struct BUserStuff	bob_BUserExt	LABEL

Flags (bob_BobFlags in ML)

name	bit	decimal	hex
SAVEBOB	0	1	0001
BOBISCOMP	1	2	0002
BWAITING	8	256	0100
BDRAWN	9	512	0200
BOBSAWAY	10	1024	0400
BOBNIX	11	2048	0800
SAVEPRESERVE	12	4096	1000
OUTSTEP	13	8192	2000

BoolInfo

10 bytes
(Updated for 3.0)

C Include File: intuition/intuition.h
ML Include File: intuition/intuition.i

byte	hex	C name	C type	ML name	ML type
0	00	Flags	UWORD	bi_Flags	WORD
2	02	Mask	UWORD *	bi_Mask	APTR
6	06	Reserved	ULONG	bi_Reserved	LONG

Flags (bi_Flags in ML)

name	bit	decimal	hex
BOOLMASK	0	1	01

BootBlock

(BB in ML)
12 bytes

C Include File: devices/bootblock.h
ML Include File: devices/bootblock.i

byte	hex	C name	C type	ML name	ML type
0	00	bb_id	UBYTE[4]	BB_ID	STRUCT
4	04	bb_chksum	LONG	BB_CHKSUM	LONG
8	08	bb_dosblock	LONG	BB_DOSBLOCK	LONG

bb_id (BB_ID in ML)

name	C value	ML value
BBID_DOS	{ 'D', 'O', 'S', '\0' }	'DOS',0
BBID_KICK	{ 'K', 'I', 'C', 'K' }	'KICK'

BootNode

20 bytes

C Include File: libraries/expansionbase.h
ML Include File: libraries/expansionbase.i

byte	hex	name	C type	ML type
0	00	bn_node	struct	Node
14	0E	bn_Flags	UWORD	UWORD
16	10	bn_DeviceNode	CPTR	CPTR

Border

16 bytes
(Updated for 3.0)

C Include File: intuition/intuition.h
ML Include File: intuition/intuition.i

byte	hex	C name	C type	ML name	ML type
0	00	LeftEdge	WORD	bd_LeftEdge	WORD
2	02	TopEdge	WORD	bd_TopEdge	WORD
4	04	FrontPen	UBYTE	bd_FrontPen	BYTE
5	05	BackPen	UBYTE	bd_BackPen	BYTE
6	06	DrawMode	UBYTE	bd_DrawMode	BYTE
7	07	Count	BYTE	bd_Count	BYTE
8	08	XY	WORD *	bd_XY	APTR
12	0C	NextBorder	struct Border *	bd_NextBorder	APTR

CardHandle

28 bytes
(New to 3.0)

C Include File: resources/card.h
ML Include File: resources/card.i

byte	hex	name	C type	ML type
0	00	cah_CardNode	struct Node	STRUCT
14	0E	cah_CardRemoved	struct Interrupt *	APTR
18	12	cah_CardInserted	struct Interrupt *	APTR
22	16	cah_CardStatus	struct Interrupt *	APTR
26	1A	cah_CardFlags	UBYTE	UBYTE

cah_CardFlags

C bit name	C flag name	ML bit name	bit	decimal	hex
CARDB_RESETREMOVE	CARDF_RESETREMOVE	CARD,RESETREMOVE	0	1	01
CARDB_IFAVAILABLE	CARDF_IFAVAILABLE	CARD,IFAVAILABLE	1	2	02
CARDB_DELAYOWNERSHIP	CARDF_DELAYOWNERSHIP	CARD,DELAYOWNERSHIP	2	4	04

CardMemoryMap

12 bytes
(New to 3.0)

C Include File: resources/card.h
ML Include File: resources/card.i

byte	hex	name	C type	ML type
0	00	cmm_CommonMemory	UBYTE *	APTR
4	04	cmm_AttributeMemory	UBYTE *	APTR
8	08	cmm_IOMemory	UBYTE *	APTR

Catalog

30 bytes
(New to 3.0)

C Include File: libraries/locale.h
ML Include File: libraries/locale.i

byte	hex	name	C type	ML type
0	00	cat_Link	struct Node	STRUCTURE
14	0E	cat_Pad	UWORD	UWORD
16	10	cat_Language	STRPTR	APTR
20	14	cat_CodeSet	ULONG	ULONG
24	18	cat_Version	UWORD	UWORD
28	1C	cat_Revision	UWORD	UWORD

CIA

3842 bytes

C Include File: hardware/cia.h
ML Include File: hardware/cia.i

byte	hex	C name	C type
0	0000	ciapra	UBYTE
1	0001	pad0	UBYTE[0xff]
256	0100	ciaprb	UBYTE
257	0101	pad1	UBYTE[0xff]
512	0200	ciaddra	UBYTE
513	0201	pad2	UBYTE[0xff]
768	0300	ciaddrb	UBYTE
769	0301	pad3	UBYTE[0xff]
1024	0400	ciaddrb	UBYTE
1025	0401	pad4	UBYTE[0xff]
1280	0500	ciatalo	UBYTE
1281	0501	pad5	UBYTE[0xff]
1536	0600	ciatahi	UBYTE
1537	0601	pad6	UBYTE[0xff]
1792	0700	ciatblo	UBYTE
1793	0701	pad7	UBYTE[0xff]
2048	0800	ciatodlo	UBYTE
2049	0801	pad8	UBYTE[0xff]
2304	0900	ciatodmid	UBYTE
2305	0901	pad9	UBYTE[0xff]
2560	0A00	ciatodhi	UBYTE

byte	hex	C name	C type
2561	0A01	pad10	UBYTE[0xff]
2816	0B00	unusedreg	UBYTE
2817	0B01	pad11	UBYTE[0xff]
3072	0C00	ciasdr	UBYTE
3073	0C01	pad12	UBYTE[0xff]
3328	0D00	ciaicr	UBYTE
3329	0D01	pad13	UBYTE[0xff]
3584	0E00	ciacra	UBYTE
3585	0E01	pad14	UBYTE[0xff]
3840	0F00	ciacrb	UBYTE

Note: The ML versions are simply equates to the numeric register offsets.

ClipboardHandle

120 bytes
(New to 2.0)

C Include File: libraries/iffparse.h
ML Include File: libraries/iffparse.i

byte	hex	name	C type	ML type
0	00	cbh_Req	struct IOClipReq	STRUCTURE
52	34	cbh_CBport	struct MsgPort	STRUCT
86	56	cbh_SatisfyPort	struct MsgPort	STRUCT

ClipboardUnitPartial

18 bytes

C Include File: devices/clipboard.h
ML Include File: devices/clipboard.i

byte	hex	name	C type	ML type
0	00	cu_Node	struct Node	STRUCT
14	0E	cu_UnitNum	ULONG	ULONG

ClipHookMsg

12 bytes
(New to 2.0)

C Include File: devices/clipboard.h
ML Include File: devices/clipboard.i

byte	hex	name	type
0	00	chm_Type	ULONG
4	04	chm_ChangeCmd	LONG
8	08	chm_ClipID	LONG

Note: chm_Type is currently 0.

ClipRect

40 bytes

C Include File: graphics/clip.h
ML Include File: graphics/clip.i

byte	hex	C name	C type	ML name	ML type
0	00	Next	struct ClipRect *	cr_Next	LONG
4	04	prev	struct Cliprect *	cr_prev	LONG
8	08	lobs	struct Layer *	cr_lobs	LONG
12	0C	BitMap	struct BitMap *	cr_BitMap	LONG
16	10	bounds	struct rectangle *	cr_MinX	WORD
18	12			cr_MinY	WORD
20	14			cr_MaxX	WORD
22	16			cr_MaxY	WORD
24	18	_p1	struct ClipRect *	cr__p1	APTR
28	1C	_p2	struct ClipRect *	cr__p2	APTR
32	20	reserved	LONG	cr_reserved	LONG
36	24	Flags	LONG	cr_Flags	LONG

CliProcList

16 bytes
(New to 3.0)

C Include File: dos/dosextens.h
ML Include File: dos/dosextens.i

byte	hex	name	C type	ML type
0	00	cpl_Node	struct MinNode	STRUCT
8	08	cpl_First	LONG	LONG
12	0C	cpl_Array	struct MsgPort **	APTR

ClockData

(CLOCKDATA in ML)
14 bytes
(New to 2.0)

C Include File: utility/date.h
ML Include File: utility/date.i

byte	hex	name	type
0	00	sec	UWORD
2	02	min	UWORD
4	04	hour	UWORD
6	06	mday	UWORD
8	08	month	UWORD
10	0A	year	UWORD
12	0C	wday	UWORD

CollectionItem

12 bytes
(Updated for 3.0)

C Include File: libraries/iffparse.h
ML Include File: libraries/iffparse.i

byte	hex	name	C type	ML type
0	00	ci_Next	struct CollectionItem *	APTR
4	04	ci_Size	LONG	LONG
8	08	ci_Data	APTR	APTR

collTable

64 bytes
(Updated for 3.0)

C Include File: graphics/gels.h
ML Include File: graphics/view.i

byte	hex	C name	C type	ML name	ML type
0	00	collPtrs	int (*collPtrs[16])()	cp_collPtrs	LONG,16

colorEntry

4 bytes
(New to 2.0)

C Include File: devices/prtgfx.h
ML Include File: devices/prtgfx.i

byte	hex	name	C type	ML type
0	00	colorLong	ULONG	LABEL
0	00	colorByte	UBYTE[4]	LABEL
0	00	colorSByte	BYTE[4]	STRUCT

ColorFontColors

8 bytes
(New to 3.0)

C Include File: graphics/text.h
ML Include File: graphics/text.i

byte	hex	name	C Type	ML type
0	00	cfc_Reserved	UWORD	UWORD
2	02	cfc_Count	UWORD	UWORD
4	04	cfc_ColorTable	UWORD *	APTR

ColorMap

52 bytes
(Updated for 3.0)

C Include File: graphics/view.h
ML Include File: graphics/view.i

byte	hex	C name	C type	ML name	ML type
0	00	Flags	UBYTE	cm_Flags	BYTE
1	01	Type	UBYTE	cm_Type	BYTE
2	02	Count	UWORD	cm_Count	WORD
4	04	ColorTable	APTR	cm_ColorTable	APTR
8	08	cm_vpe	struct ViewPortExtra *	cm_vpe	APTR
12	0C	TransparencyBits	UWORD *	cm_TransparencyBits	APTR
16	10	TransparencyPlane	UBYTE	cm_TransparencyPlane	BYTE
17	11	reserved1	UBYTE	cm_reserved1	BYTE
18	12	reserved2	UWORD	cm_reserved2	WORD

byte	hex	C name	C type	ML name	ML type
20	14	cm_vp	struct ViewPort *	cm_vp	APTR
24	18	NormalDisplayInfo	APTR	cm_NormalDisplayInfo	APTR
28	1C	CoerceDisplayInfo	APTR	cm_CoerceDisplayInfo	APTR
32	20	cm_batch_items	struct TagItem *	cm_batch_items	APTR
36	24	VPModeID	ULONG	cm_VPModeID	ULONG
40	28	PalExtra	struct PaletteExtra *	cm_PalExtra	APTR
44	2C	SpriteBase_Even	UWORD	cm_SpriteBase_Even	UWORD
46	2E	SpriteBase_Odd	UWORD	cm_SpriteBase_Odd	UWORD
48	30	Bp_0_base	UWORD	cm_Bp_0_base	UWORD
50	32	Bp_1_base	UWORD	cm_Bp_1_base	UWORD

Flags (cm_Flags in ML)

name	bit	decimal	hex
COLORMAP_TRANSPARENCY	0	1	01
COLORPLANE_TRANSPARENCY	1	2	02
BORDER_BLANKING	2	4	04
BORDER_NOTRANSPARENCY	3	8	08
VIDEOCONTROL_BATCH	4	16	10
USER_COPPER_CLIP	5	32	20
BORDERSPRITES	6	64	40

Type (cm_Type in ML)

name	decimal	hex
COLORMAP_TYPE_V1_2	0	00
COLORMAP_TYPE_V1_4	1	01
COLORMAP_TYPE_V36	1	01
COLORMAP_TYPE_V39	2	02

ColorRegister

3 bytes
(New to 3.0)

C Include File: datatypes/pictureclass.h
ML Include File: datatypes/pictureclass.i

byte	hex	name	type
0	00	red	UBYTE
1	01	green	UBYTE
2	02	blue	UBYTE

ColorSpec

8 bytes
(New to 2.0)

C Include File: intuition/intuition.h
ML Include File: intuition/intuition.i

byte	hex	C name	C type	ML name	ML type
0	00	ColorIndex	WORD	cs_ColorIndex	WORD
2	02	Red	UWORD	cs_Red	UWORD
4	04	Green	UWORD	cs_Green	UWORD
6	06	Blue	UWORD	cs_Blue	UWORD

Note: The size of this structure is variable. A -1 in the ColorIndex field terminates an array of of ColorSpecs.

ColorTextFont

96 bytes
(New to 2.0)

C Include File: graphics/text.h
ML Include File: graphics/text.i

byte	hex	name	C type	ML type
0	00	ctf_TF	struct TextFont	STRUCTURE
52	34	ctf_Flags	UWORD	UWORD
54	36	ctf_Depth	UBYTE	UBYTE
55	37	ctf_FgColor	UBYTE	UBYTE
56	38	ctf_Low	UBYTE	UBYTE
57	39	ctf_High	UBYTE	UBYTE
58	3A	ctf_PlanePick	UBYTE	UBYTE
59	3B	ctf_PlaneOnOff	UBYTE	UBYTE
60	3C	ctf_ColorFontColors	struct ColorFontColors *	APTR
64	40	ctf_CharData	APTR[8]	STRUCT

ctf_Flags

name	bit	decimal	hex
CT_COLORFONT	0	1	01
CT_GREYFONT	1	2	02
CT_ANTIALIAS	2	4	04

ColorWheelHSB

12 bytes
(New to 3.0)

C Include File: gadgets/colorwheel.h
ML Include File: gadgets/colorwheel.i

byte	hex	name	type
0	00	cw_Hue	ULONG
4	04	cw_Saturation	ULONG
8	08	cw_Brightness	ULONG

ColorWheelRGB

12 bytes
(New to 3.0)

C Include File: gadgets/colorwheel.h
ML Include File: gadgets/colorwheel.i

byte	hex	name	type
0	00	cw_Red	ULONG
4	04	cw_Green	ULONG
8	08	cw_Blue	ULONG

CommandLineInterface

64 bytes
(Updated for 3.0)

C Include File: dos/dosextens.h
ML Include File: dos/dosextens.i

byte	hex	name	type
0	00	cli_Result2	LONG
4	04	cli_SetName	BSTR
8	08	cli_CommandDir	BPTR
12	0C	cli_ReturnCode	LONG
16	10	cli_CommandName	BSTR
20	14	cli_FailLevel	LONG
24	18	cli_Prompt	BSTR
28	1C	cli_StandardInput	BPTR
32	20	cli_CurrentInput	BPTR

byte	hex	name	type
36	24	cli_CommandFile	BSTR
40	28	cli_Interactive	LONG
44	2C	cli_Background	LONG
48	30	cli_CurrentOutput	BPTR
52	34	cli_DefaultStack	LONG
56	38	cli_StandardOutput	BPTR
60	3C	cli_Module	BPTR

Note: This structure was previously in libraries/dosextens.h (C version) and libraries/dosextens.i (ML version).

ConfigDev

68 bytes

C Include File: libraries/configvars.h
ML Include File: libraries/configvars.i

byte	hex	name	C type	ML type
0	00	cd_Node	Node	STRUCT
14	0E	cd_Flags	UBYTE	UBYTE
15	0F	cd_Pad	UBYTE	UBYTE
16	10	cd_Rom	struct ExpansionRom	STRUCT
32	20	cd_BoardAddr	APTR	APTR
36	24	cd_BoardSize	APTR	APTR
40	28	cd_SlotAddr	UWORD	UWORD
42	2A	cd_SlotSize	UWORD	UWORD
44	2C	cd_Driver	APTR	APTR
48	30	cd_NextCD	struct ConfigDev *	APTR
52	34	cd_Unused	ULONG[4]	STRUCT

cd_Flags

C flag name	C bit name	ML bit name	bit	decimal	hex
CDF_SHUTUP	CDB_SHUTUP	CD,SHUTUP	0	1	01
CDF_CONFIGME	CDB_CONFIGME	CD,CONFIGME	1	2	02
CDF_BADMEMORY	CDB_BADMEMORY	CD,BADMEMORY	2	4	04
CDF_PROCESSED	CDB_PROCESSED	CD,PROCESSED	3	8	08

ContextNode

24 bytes
(New to 2.0)

C Include File: libraries/iffparse.h
ML Include File: libraries/iffparse.i

byte	hex	name	C type	ML type
0	00	cn_Node	struct MinNode	STRUCTURE
8	08	cn_ID	LONG	LONG
12	0C	cn_Type	LONG	LONG
16	10	cn_Size	LONG	LONG
20	14	cn_Scan	LONG	LONG

ConUnit

296 bytes

C Include File: devices/conunit.h
ML Include File: devices/conunit.i

byte	hex	name	C type	ML type
0	0000	cu_MP	struct MsgPort	STRUCTURE
34	0022	cu_Window	struct Window *	APTR
38	0026	cu_XCP	WORD	WORD
40	0028	cu_YCP	WORD	WORD
42	002A	cu_XMax	WORD	WORD
44	002C	cu_YMax	WORD	WORD
46	002E	cu_XRSize	WORD	WORD
48	0030	cu_YRSize	WORD	WORD
50	0032	cu_XROrigin	WORD	WORD
52	0034	cu_YROrigin	WORD	WORD
54	0036	cu_XRExtant	WORD	WORD
56	0038	cu_YRExtant	WORD	WORD
58	003A	cu_XMinShrink	WORD	WORD
60	003C	cu_YMinShrink	WORD	WORD
62	003E	cu_XCCP	WORD	WORD
64	0040	cu_YCCP	WORD	WORD
66	0042	cu_KeyMapStruct	struct KeyMap	STRUCT
98	0062	cu_TabStops	UWORD[80]	STRUCT
258	0102	cu_Mask	byte	BYTE
259	0103	cu_FgPen	byte	BYTE
260	0104	cu_BgPen	byte	BYTE
261	0105	cu_AOLPen	byte	BYTE

byte	hex	name	C type	ML type
262	0106	cu_DrawMode	byte	BYTE
263	0107	cu_AreaPtSize	byte	BYTE
264	0108	cu_AreaPtrn	APTR	APTR
268	010C	cu_Minterms	UBYTE[8]	STRUCT
276	0114	cu_Font	structTextFont *	APTR
280	0118	cu_AlgoStyle	UBYTE	UBYTE
281	0119	cu_TxFlags	UBYTE	UBYTE
282	011A	cu_TxHeight	UWORD	UWORD
284	011C	cu_TxWidth	UWORD	UWORD
286	011E	cu_TxBaseline	UWORD	UWORD
288	0120	cu_TxSpacing	UWORD	UWORD
290	0122	cu_Modes	UBYTE[3]	STRUCT
293	0125	cu_RawEvents	UBYTE[3]	STRUCT

cu_algoStyle

name	bit	decimal	hex
FSF_UNDERLINED	0	1	01
FSF_BOLD	1	2	02
FSF_ITALIC	2	4	04
FSF_EXTENDED	3	8	08

cu_TxFlags

name	bit	decimal	hex
FPF_ROMFONT	0	1	01
FPF_DISKFONT	1	2	02
FPF_REVPATH	2	4	04
FPF_TALLDOT	3	8	08
FPF_LONGDOT	4	16	10
FPF_PROPORTIONAL	5	32	20
FPF_DESIGNED	6	64	40
FPF_REMOVED	7	128	80

copinit

140 bytes
(Updated for 3.0)

C Include File: graphics/copper.h
ML Include File: graphics/copper.i

byte	hex	C name	C type	ML name	ML type
0	00	vsync_hblank	UWORD[2]	copinit_vsync_hblank	STRUCT
4	04	diagstrt	UWORD[12]	copinit_diagstrt	STRUCT
28	1C	fm0	UWORD[2]	copinit_fm0	STRUCT

byte	hex	C name	C type	ML name	ML type
32	20	diwstart	UWORD[10]	copinit_diwstart	STRUCT
52	34	bplcon2	UWORD[2]	copinit_bplcon2	STRUCT
56	38	sprfix	UWORD[2*8]	copinit_sprfix	STRUCT
72	48	sprstrtup	UWORD[(2*8*2)]	copinit_sprstrtup	STRUCT
104	68	wait14	UWORD[2]	copinit_wait14	STRUCT
108	6C	norm_hblank	UWORD[2]	copinit_genloc	STRUCT
112	70	jump	UWORD[(2*2)]		
120	78	wait_forever	UWORD[2]		
124	7C	sprstop	UWORD[8]	copinit_sprstop	STRUCT

CopIns

6 bytes

C Include File: graphics/copper.h
ML Include File: graphics/copper.i

byte	hex	C name	C type	ML name	ML type
0	00	OpCode	WORD	ci_OpCode	WORD
2	02	nxtlist	struct CopList *	ci_nxtlist	STRUCT
2	02	VWaitPos	WORD	ci_VWaitPos	STRUCT
2	02	DestAddr	WORD	ci_DestAddr	STRUCT
4	04	HWaitPos	WORD	ci_HWaitPos	STRUCT
4	04	DestData	WORD	ci_DestData	STRUCT

CopList

50 bytes
(Updated for 3.0)

C Include File: graphics/copper.h
ML Include File: graphics/copper.i

byte	hex	C name	C type	ML name	ML type
0	00	Next	struct CopList *	cl_Next	APTR
4	04	_CopList	struct CopList *	cl__CopList	APTR
8	08	_ViewPort	struct ViewPort *	cl__ViewPort	APTR
12	0C	CopIns	struct Copins *	cl_CopIns	APTR
16	10	CopPtr	struct CopPtr *	cl_CopPtr	APTR
20	14	CopLStart	UWORD *	cl_CopLStart	APTR
24	18	CopSStart	UWORD *	cl_CopSStart	APTR
28	1C	Count	WORD	cl_Count	WORD
30	1E	MaxCount	WORD	cl_MaxCount	WORD
32	20	DyOffset	WORD	cl_DyOffset	WORD

byte	hex	C name	C type	ML name	ML type
34	22	Cop2Start	UWORD *	cl_Cop2Start	APTR
38	26	Cop3Start	UWORD *	cl_Cop3Start	APTR
42	2A	Cop4Start	UWORD *	cl_Cop4Start	APTR
46	2E	Cop5Start	UWORD *	cl_Cop5Start	APTR

CountryPrefs

498 bytes
(New to 3.0)

C Include File: prefs/locale.h
ML Include File: prefs/locale.i

byte	hex	name	C type	ML type
0	0000	cp_Reserved	ULONG[4]	STRUCT
16	0010	cp_CountryCode	ULONG	ULONG
20	0014	cp_TelephoneCode	ULONG	ULONG
24	0018	cp_MeasuringSystem	UBYTE	UBYTE
26	001A	cp_DateTimeFormat	char[80]	STRUCT
106	006A	cp_DateFormat	char[40]	STRUCT
146	0092	cp_TimeFormat	char[40]	STRUCT
186	00BA	cp_ShortDateTimeFormat	char[80]	STRUCT
266	010A	cp_ShortDateFormat	char[40]	STRUCT
306	0132	cp_ShortTimeFormat	char[40]	STRUCT
346	015A	cp_DecimalPoint	char[10]	STRUCT
356	0164	cp_GroupSeparator	char[10]	STRUCT
366	016E	cp_FracGroupSeparator	char[10]	STRUCT
376	0178	cp_Grouping	UBYTE	UBYTE
377	0179	cp_FracGrouping	UBYTE	UBYTE
378	017A	cp_MonDecimalPoint	char[10]	STRUCT
388	0184	cp_MonGroupSeparator	char[10]	STRUCT
398	018E	cp_MonFracGroupSeparator	char[10]	STRUCT
408	0198	cp_MonGrouping	UBYTE[10]	STRUCT
418	01A2	cp_MonFracGrouping	UBYTE[10]	STRUCT
428	01AC	cp_MonFracDigits	UBYTE	UBYTE
429	01AD	cp_MonIntFracDigits	UBYTE	UBYTE
430	01AE	cp_MonCS	char[10]	STRUCT
440	01B8	cp_MonSmallCS	char[10]	STRUCT
450	01C2	cp_MonIntCS	char[10]	STRUCT
460	01CC	cp_MonPositiveSign	char[10]	STRUCT
470	01D6	cp_MonPositiveSpaceSep	UBYTE	UBYTE
471	01D7	cp_MonPositiveSignPos	UBYTE	UBYTE
472	01D8	cp_MonPositiveCSPos	UBYTE	UBYTE
474	01DA	cp_MonNegativeSign	char[10]	STRUCT

byte	hex	name	C type	ML type
484	01E4	cp_MonNegativeSpaceSep	UBYTE	UBYTE
495	01E5	cp_MonNegativeSignPos	UBYTE	UBYTE
496	01E6	cp_MonNegativeCSPos	UBYTE	UBYTE
497	01E7	cp_CalendarType	UBYTE	UBYTE

CSource

12 bytes
(New to 2.0)

C Include File: dos/rdargs.h
ML Include File: dos/rdargs.i

byte	hex	name	C type	ML type
0	00	CS_Buffer	UBYTE *	APTR
4	04	CS_Length	LONG	LONG
8	08	CS_CurChr	LONG	LONG

CurrentBinding

16 bytes

C Include File: libraries/configvars.h
ML Include File: libraries/configvars.i

byte	hex	name	C type	ML type
0	00	cb_ConfigDev	struct ConfigDev *	APTR
4	04	cb_FileName	UBYTE *	APTR
8	08	cb_ProductString	UBYTE *	APTR
12	0C	cb_ToolTypes	UBYTE **	APTR

Custom

486 bytes
(Expanded for 2.0)

C Include File: hardware/custom.h
ML Include File: hardware/custom.i

byte	hex	name	C type	ML type
0	0000	bltddat	UWORD	$00
2	0002	dmaconr	UWORD	$02
4	0004	vposr	BITSET	$04
6	0006	vhposr	UWORD	$06
8	0008	dskdatr	UWORD	$08
10	000A	joy0dat	UWORD	$0A
12	000C	joy1dat	UWORD	$0C
14	000E	clxdat	UWORD	$0E
16	0010	adkconr	UWORD	$10
18	0012	pot0dat	UWORD	$12
20	0014	pot1dat	UWORD	$14
22	0016	potinp	UWORD	$16
24	0018	serdatr	UWORD	$18
26	001A	dskbytr	UWORD	$1A
28	001C	intenar	UWORD	$1C
30	001E	intreqr	UWORD	$1E
32	0020	dskpt	APTR	$20
36	0024	dsklen	UWORD	$24
38	0026	dskdat	UWORD	$26
40	0028	refptr	UWORD	$28
42	002A	vposw	UWORD	$2A
44	002C	vhposw	UWORD	$2C
46	002E	copcon	UWORD	$2E
48	0030	serdat	UWORD	$30
50	0032	serper	UWORD	$32
52	0034	potgo	UWORD	$34
54	0036	joytest	UWORD	$36
56	0038	strequ	UWORD	$38
58	003A	strvbl	UWORD	$3A
60	003C	strhor	UWORD	$3C
62	003E	strlong	UWORD	$3E
64	0040	bltcon0	UWORD	$40
66	0042	bltcon1	UWORD	$42
68	0044	bltafwm	UWORD	$44
70	0046	bltalwm	UWORD	$46
72	0048	bltcpt	APTR	$48
76	004C	bltbpt	APTR	$4C
80	0050	bltapt	APTR	$50
84	0054	bltdpt	APTR	$54
88	0058	bltsize	UWORD	$58
90	005A	pad2d	UWORD[3]	
96	0060	bltcmod	UWORD	$60
98	0062	bltbmod	UWORD	$62
100	0064	bltamod	UWORD	$64

byte	hex	name	C type	ML type
102	0066	bltdmod	UWORD	$66
104	0068	pad34	UWORD[4]	
112	0070	bltcdat	UWORD	$70
114	0072	bltbdat	UWORD	$72
116	0074	bltadat	UWORD	$74
118	0076	pad3b	UWORD[4]	
126	007E	dsksync	UWORD	$7E
128	0080	cop1lc	ULONG	$80
132	0084	cop2lc	ULONG	$84
136	0088	copjmp1	UWORD	$88
138	008A	copjmp2	UWORD	$8A
140	008C	copins	UWORD	$8C
142	008E	diwstrt	UWORD	$8E
144	0090	diwstop	UWORD	$90
146	0092	ddfstrt	UWORD	$92
148	0094	ddfstop	UWORD	$94
150	0096	dmacon	UWORD	$96
152	0098	clxcon	UWORD	$98
154	009A	intena	UWORD	$9A
156	009C	intreq	UWORD	$9C
158	009E	adkcon	UWORD	$9E
160	00A0	aud	struct AudChannel[4]	STRUCTURE
224	00A2	bplpt	APTR[6]	$E0
256	0100	bplcon0	UWORD	$100
258	0102	bplcon1	UWORD	$102
260	0104	bplcon2	UWORD	$104
262	0106	bplcon3	UWORD	$106
264	0108	bpl1mod	UWORD	$108
266	010A	bpl2mod	UWORD	$10A
268	010C	pad86	UWORD[2]	
272	0110	bpldat	UWORD[6]	$110
284	011C	pad8e	UWORD[2]	
288	0120	sprpt	APTR[8]	$120
320	0140	spr	struct SpriteDef[8]	$140
384	0180	color	UWORD[32]	$180
448	01C0	htotal	UWORD	$1C0
450	01C2	hsstop	UWORD	$1C2
452	01C4	hbstrt	UWORD	$1C4
454	01C6	hbstop	UWORD	$1C6
464	01C8	vtotal	UWORD	$1C8
466	01CA	vsstop	UWORD	$1CA
468	01CC	vbstrt	UWORD	$1CC
470	01CE	vbstop	UWORD	$1CA

byte	hex	name	C type	ML type
472	01D0	sprhstrt	UWORD	$1D0
474	01D2	sprhstnop	UWORD	$1D2
476	01D4	bplhstrt	UWORD	$1D4
478	01D6	bplhstnop	UWORD	$1D6
480	01D8	hhposw	UWORD	$1D8
482	01DA	hhposr	UWORD	$1DA
484	01DC	beamcon0	UWORD	$1DC
486	01DE	hsstrt	UWORD	$1DE
488	01E0	vsstrt	UWORD	$1E0
490	01E2	hcenter	UWORD	$1E2
492	01E4	diwhigh	UWORD	$1E4
494	01E6	padf3[11]	UWORD	$1E6
516	0204	fmode	UWORD	$1FC

Note: The ML and C definitions of fmode do not agree.

bltcon0

C name	ML name	bit	decimal	hex
NANBNC	NANBNC	0	1	0001
NANBC	NANBC	1	2	0002
NABNC	NABNC	2	4	0004
NABC	NABC	3	8	0008
ANBNC	ANBNC	4	16	0010
ANBC	ANBC	5	32	0020
ABNC	ABNC	6	64	0040
ABC	ABC	7	128	0080
BC0B_DEST	BC0BDest	8	256	0100
BC0B_SRCC	BC0BSrcC	9	512	0200
BC0B_SRCB	B0CBSrcB	10	1024	0400
BC0B_SRCA	B0BCSrcA	11	2048	0800

bltcon1

name	bit	decimal	hex
LINEMODE	0	1	01
ONEDOT	1	2	02
FILL_CARRYIN	2	4	04
FILL_O	3	8	08
FILL_XOR	4	16	10
OVFLAG	5	32	20
SIGNFLAG	6	64	40

beamcon0

name	bit	decimal	hex
HSYNCTRUE	0	1	0001
VSYNCTRUE	1	2	0002
CSYNCTRUE	2	4	0004
CSBLANK	3	8	0008
VARCSYNC	4	16	0010
DISPLAYPAL	5	32	0020
DISPLAYDUAL	6	64	0040
VARBEAM	7	128	0080
VARHSYNC	8	256	0100
VARVSYNC	9	512	0200
CSCBLANKEN	10	1024	0400
LOLDIS	11	2048	0800
VARVBLANK	12	4096	1000

bplcon2

name	bit	decimal	hex
BPLCON2_ZDCTEN	10	1024	0400
BPLCON2_ZDBPEN	11	2048	0800
BPLCON2_ZDBPSEL0	12	4096	1000
BPLCON2_ZDBPSEL1	13	8192	2000
BPLCON2_ZDBPSEL2	14	16384	4000

bplcon3

name	bit	decimal	hex
BPLCON3_EXTBLNKEN	0	1	01
BPLCON3_EXTBLKZD	1	2	02
BPLCON3_ZDCLKEN	2	4	04
BPLCON3_BRDNTRAN	4	16	10
BPLCON3_BRDNBLNK	5	32	20

cprlist

10 bytes
(Updated for 3.0)

C Include File: graphics/copper.h
ML Include File: graphics/copper.i

byte	hex	C name	C typ	ML name	ML type
0	00	Next	struct cprlist *	crl_Next	APTR
4	04	start	UWORD *	crl_start	APTR
8	08	MaxCount	WORD	crl_MaxCount	WORD

DataType

58 bytes
(New to 3.0)

C Include File: datatypes/datatypes.h
ML Include File: datatypes/datatypes.i

byte	hex	name	C type	ML type
0	00	dtn_Node1	struct Node	STRUCT
14	0E	dtn_Node2	struct Node	STRUCT
28	1C	dtn_Header	struct DataTypeHeader *	APTR
32	20	dtn_ToolList	struct List	STRUCT
46	2E	dtn_FunctionName	STRPTR	APTR
50	32	dtn_AttrList	struct TagItem *	APTR
54	36	dtn_Length	ULONG	ULONG

DataTypeHeader

32 bytes
(New to 3.0)

C Include File: datatypes/datatypes.h
ML Include File: datatypes/datatypes.i

byte	hex	name	C type	ML type
0	00	dth_Name	STRPTR	ULONG
4	04	dth_BaseName	STRPTR	ULONG
8	08	dth_Pattern	STRPTR	ULONG
12	0C	dth_Mask	WORD *	APTR
16	10	dth_GroupID	ULONG	ULONG
20	14	dth_ID	ULONG	ULONG
24	18	dth_MaskLen	WORD	WORD
26	1A	dth_Pad	WORD	WORD
28	1C	dth_Flags	UWORD	UWORD
30	1E	dth_Priority	UWORD	UWORD

dth_Flags

flags name	decimal	hex
DTF_BINARY	0	0000
DTF_ASCII	1	0001
DTF_IFF	2	0002
DTF_MISC	3	0003
DTF_TYPE_MASK	15	000F
DTF_CASE	16	0010
DTF_SYSTEM1	4096	1000

DateStamp

12 bytes
(Updated for 3.0)

C Include File: dos/dos.h
ML Include File: dos/dos.i

byte	hex	name	type
0	00	ds_Days	LONG
4	04	ds_Minute	LONG
8	08	ds_Tick	LONG

Note: This structure was previously found in libraries/dos.h (C version) and libraries/dos.i (ML version).

DateTime

26 bytes
(New to 2.0)

C Include File: dos/datetime.h
ML Include File: dos/datetime.i

byte	hex	name	C type	ML type
0	00	dat_Stamp	struct DateStamp	STRUCT
12	0C	dat_Format	UBYTE	UBYTE
13	0D	dat_Flags	UBYTE	UBYTE
14	0E	dat_StrDay	UBYTE *	CPTR
18	12	dat_StrDate	UBYTE *	CPTR
22	16	dat_StrTime	UBYTE *	CPTR

dat_Flags

C flags name	C bit name	ML bit name	bit	decimal	hex
DTF_SUBST	DFB_SUBST	DT,SUBST	0	1	01
DTF_FUTURE	DFB_FUTURE	DT,FUTURE	1	2	02

DBP

see DBufPacket

DBufInfo

84 bytes
(New to 3.0)

C Include File: graphics/view.h
ML Include File: graphics/view.i

byte	hex	name	C type	ML type
0	00	dbi_Link1	APTR	APTR
4	04	dbi_Count1	ULONG	ULONG
8	08	dbi_SafeMessage	struct Message	STRUCT
28	1C	dbi_UserData1	APTR	APTR
32	20	dbi_Link2	APTR	APTR
36	24	dbi_Count2	ULONG	ULONG
40	28	dbi_DispMessage	struct Message	STRUCT
60	3C	dbi_UserData2	APTR	APTR
64	40	dbi_MatchLong	ULONG	ULONG
68	44	dbi_CopPtr1	APTR	APTR
72	48	dbi_CopPtr2	APTR	APTR
76	4C	dbi_CopPtr3	APTR	APTR
80	50	dbi_BeamPos1	UWORD	UWORD
82	52	dbi_BeamPos2	UWORD	UWORD

DBufPacket

(DBP in ML)
12 bytes

C Include File: graphics/gels.h
ML Include File: graphics/gels.i

byte	hex	C name	C type	ML name	ML type
0	00	BufY	WORD	dbp_BufY	WORD
2	02	BufX	WORD	dbp_BufX	WORD
4	04	BufPath	struct VSprite *	dbp_BufPath	APTR
8	08	BufBuffer	WORD *	dbp_BufBuffer	APTR

DD

see Device

Device

(DD in ML)
34 bytes

C Include File: exec/devices.h
ML Include File: exec/devices.i

byte	hex	name	C type	ML type
0	00	dd_Library	Library	STRUCTURE

DeviceData

52 bytes

C Include File: devices/prtbase.h
ML Include File: devices/prtbase.i

byte	hex	name	C type	ML type
0	00	dd_Device	struct Library	
34	22	dd_Segment	APTR	APTR
38	26	dd_ExecBase	APTR	APTR
42	2A	dd_CmdVectors	APTR	APTR
46	2E	dd_CmdBytes	APTR	APTR
50	32	dd_NumCommands	UWORD	UWORD

DeviceList

(DevList in ML)
44 bytes
(Updated for 3.0)

C Include File: dos/dosextens.h
ML Include File: dos/dosextens.i

byte	hex	name	C type	ML type
0	00	dl_Next	BPTR	BPTR
4	04	dl_Type	LONG	LONG
8	08	dl_Task	struct MsgPort *	APTR
12	0C	dl_Lock	BPTR	BPTR
16	10	dl_VolumeDate	struct DateStamp	STRUCT
28	1C	dl_LockList	BPTR	BPTR
32	20	dl_DiskType	LONG	LONG
36	24	dl_unused	LONG	LONG
40	28	dl_Name	BSTR	BSTR

dl_Type

name	bit	decimal	hex
DLT_DEVICE	0	1	01
DLT_DIRECTORY	1	2	02
DLT_VOLUME	2	4	04
DLT_LATE	3	8	08
DLT_NONBINDING	4	16	10

Note: This structure was previously located in libraries/dosextens.h (C version) and libraries/dosextens.i (ML version).

DeviceNode

44 bytes

C Include File: dos/filehandler.h
ML Include File: dos/filehandler.i

byte	hex	name	C type	ML type
0	00	dn_Next	BPTR	BPTR
4	04	dn_Type	ULONG	ULONG
8	08	dn_Task	struct MsgPort *	CPTR
12	0C	dn_Lock	BPTR	BPTR
16	10	dn_Handler	BSTR	BSTR
20	14	dn_StackSize	ULONG	ULONG
24	18	dn_Priority	LONG	LONG
28	1C	dn_Startup	BPTR	BPTR
32	20	dn_SegList	BPTR	BPTR
36	24	dn_GlobalVec	BPTR	BPTR
40	28	dn_Name	BSTR	BSTR

Note: dn_Type is currently always 0. This structure was previously located in libraries/filehandler.h (C version) and libraries/filehandler.i (ML version).

DeviceTData

10 bytes
(New to 3.0)

C Include File: resources/card.h
ML Include File: resources/card.i

byte	hex	name	type
0	00	dtd_DTsize	ULONG
4	04	dtd_DTspeed	ULONG
8	08	dtd_DTtype	UBYTE
9	09	dtd_DTflags	UBYTE

DevInfo

44 bytes
(Updated for 3.0)

C Include File: dos/dosextens.h
ML Include File: dos/dosextens.i

byte	hex	name	type
0	00	dvi_Next	BPTR
4	04	dvi_Type	LONG
8	08	dvi_Task	APTR
12	0C	dvi_Lock	BPTR
16	10	dvi_Handler	BSTR
20	14	dvi_StackSize	LONG
24	18	dvi_Priority	LONG
28	1C	dvi_Startup	LONG
32	20	dvi_SegList	BPTR
36	24	dvi_GlobalVec	BPTR
40	28	dvi_Name	BSTR

Note: dvi_Type is currently always 0.

DevList

see DeviceList

DevProc

16 bytes

C Include File: dos/dosextens.h
ML Include File: dos/dosextens.i

byte	hex	name	C type	ML type
0	00	dvp_Port	struct MsgPort *	APTR
4	04	dvp_Lock	BPTR	BPTR
8	08	dvp_Flags	ULONG	ULONG
12	0C	dvp_DevNode	struct DosList*	APTR

dvp_Flags

C flag name	C bit name	ML bit name	bit	decimal	hex
DVPF_UNLOCK	DVPB_UNLOCK	DVP,UNLOCK	0	1	01
DVPF_ASSIGN	DVPB_ASSIGN	DVP,ASSIGN	1	2	02

DiagArea

14 bytes

C Include File: libraries/configregs.h
ML Include File: libraries/configregs.i

byte	hex	name	type
0	00	da_Config	UBYTE
1	01	da_Flags	UBYTE
2	02	da_Size	UWORD
4	04	da_DiagPoint	UWORD
6	06	da_BootPoint	UWORD
8	08	da_Name	UWORD
10	0A	da_Reserved01	UWORD
12	0C	da_Reserved02	UWORD

da_Config

name	decimal	hex
DAC_NEVER	0	00
DAC_NIBBLEWIDE	0	00
DAC_CONFIGTIME	16	10
DAC_BINDTIME	32	20
DAC_BOOTTIME	48	30
DAC_BYTEWIDE	64	40
DAC_WORDWIDE	128	80

DimensionInfo

88 bytes
(New to 2.0)

C Include File: graphics/displayinfo.h
ML Include File: graphics/displayinfo.i

byte	hex	name	C type	ML name	ML type
0	00	Header	QueryHeader	dim_Header	STRUCTURE
16	10	MaxDepth	UWORD	dim_MaxDepth	UWORD
18	12	MinRasterWidth	UWORD	dim_MinRasterWidth	UWORD

byte	hex	name	C type	ML name	ML type
20	14	MinRasterHeight	UWORD	dim_MinRasterHeight	UWORD
22	16	MaxRasterWidth	UWORD	dim_MaxRasterWidth	UWORD
24	18	MaxRasterHeight	UWORD	dim_MaxRasterHeight	UWORD
26	1A	Nominal	struct Rectangle	dim_Nominal	STRUCT
34	22	MaxOScan	struct Rectangle	dim_MaxOScan	STRUCT
42	2A	VideoOscan	struct Rectangle	dim_VideoOScan	STRUCT
50	32	TxtOScan	struct Rectangle	dim_TxtOScan	STRUCT
58	3A	StdOScan	struct Rectangle	dim_StdOScan	STRUCT
66	42	pad	UBYTE[14]	dim_Pad	STRUCT
80	50	reserved	ULONG[2]	dim_reserved	STRUCT

DiscResource

(DISCRESOURCE in ML)
148 bytes
(Updated for 2.0)

C Include File: resources/disk.h
ML Include File: resources/disk.i

byte	hex	C name	C type	ML name	ML type
0	00	dr_Library	struct Library	DR_LIBRARY	STRUCTURE
34	22	dr_Current	struct DiscResourcePtr	DR_CURRENT	APTR
38	26	dr_Flags	UBYTE	DR_FLAGS	UBYTE
39	27	dr_pad	UBYTE	DR_PAD	UBYTE
40	28	dr_SysLib	struct Library *	DR_SYSLIB	APTR
44	2C	dr_CiaResource	struct Library *	DR_CIARESOURCE	APTR
48	30	dr_UnitID	ULONG[4]	DR_UNITID	STRUCT
64	40	dr_Waiting	struct List	DR_WAITING	STRUCT
78	4E	dr_DiskBlock	struct Interrupt	DR_DISKBLOCK	STRUCT
100	64	dr_DiscSync	struct Interrupt	DR_DISCSYNC	STRUCT
122	7A	dr_Index	struct Interrupt	DR_INDEX	STRUCT
144	90	dr_CurrTask	struct Task *	DR_CURRTASK	APTR

dr_Flags (DR_FLAGS in ML)

C flag name	C bit name	ML name	bit	decimal	hex
DRF_ALLOC0	DRB_ALLOC0	DR,ALLOC0	0	1	01
DRF_ALLOC1	DRB_ALLOC1	DR,ALLOC1	1	2	02
DRF_ALLOC2	DRB_ALLOC2	DR,ALLOC2	2	4	04
DRF_ALLOC3	DRB_ALLOC3	DR,ALLOC3	3	8	08
DRF_ACTIVE	DRB_ACTIVE	DR,ACTIVE	7	128	80

DiscResourceUnit

86 bytes

C Include File: resources/disk.h
ML Include File: resources/disk.i

byte	hex	C name	C type	ML name	ML type
0	00	dru_Message	struct Message	DRU_MESSAGE	STRUCTURE
20	14	dru_DiscBlock	struct Interrupt	DRU_DISCBLOCK	STRUCT
42	2A	dru_DiscSync	struct Interrupt	DRU_DISCSYNC	STRUCT
64	40	dru_Index	struct Interrupt	DRU_INDEX	STRUCT

DiskFontHeader

106 bytes
(New to 2.0)

C Include File: diskfont/diskfont.h
ML Include File: diskfont/diskfont.i

byte	hex	name	C type	ML type
0	00	dfh_DF	struct Node	STRUCT
14	0E	dfh_FileID	UWORD	UWORD
16	10	dfh_Revision	UWORD	UWORD
18	12	dfh_Segment	LONG	LONG
22	16	dfh_Name	char[32]	STRUCT
54	36	dfh_TF	struct TextFont	STRUCT

Note: This structure was previously located in libraries/diskfont.h (C version) and libraries/diskfont.i (ML version).

DiskObject

78 bytes

C Include File: workbench/workbench.h
ML Include File: workbench/workbench.i

byte	hex	name	C type	ML type
0	00	do_Magic	UWORD	UWORD
2	02	do_Version	UWORD	UWORD
4	04	do_Gadget	struct Gadget	STRUCT
48	30	do_Type	UBYTE	UBYTE

byte	hex	name	C type	ML type
50	32	do_DefaultTool	char *	APTR
54	36	do_ToolTypes	char **	APTR
58	3A	do_CurrentX	LONG	LONG
62	3E	do_CurrentY	LONG	LONG
66	42	do_DrawerData	struct DrawerData *	APTR
70	46	do_ToolWindow	char *	APTR
74	4A	do_StackSize	LONG	LONG

Note: do_Magic is $E310.

DisplayInfo

56 bytes
(Updated for 3.0)

C Include File: graphics/displayinfo.h
ML Include File: graphics/displayinfo.i

byte	hex	C name	C type	ML name	ML type
0	00	Header	struct QueryHeader	dis_Header	STRUCTURE
16	10	NotAvailable	UWORD	dis_NotAvailable	UWORD
18	12	PropertyFlags	ULONG	dis_PropertyFlags	ULONG
22	16	Resolution	Point	dis_Resolution	STRUCT
26	1A	PixelSpeed	UWORD	dis_PixelSpeed	UWORD
28	1C	NumStdSprites	UWORD	dis_NumStdSprites	UWORD
30	1E	PaletteRange	UWORD	dis_PaletteRange	UWORD
32	20	SpriteResolution	Point	dis_SpriteResolution	STRUCT
36	24	pad	UBYTE[4]	dis_pad	STRUCT
40	28	RedBits	UBYTE	RedBits	UBYTE
41	29	GreenBits	UBYTE	GreenBits	UBYTE
42	2A	BlueBits	UBYTE	BlueBits	UBYTE
43	2B	pad2	UBYTE[5]	dis_pad2	STRUCT
48	30	reserved	ULONG[2]	dis_reserved	STRUCT

DisplayMode

106 bytes
(New to 3.0)

C Include File: libraries/asl.h
ML Include File: libraries/asl.i

byte	hex	name	C type	ML name	ML type
0	00	dm_Node	struct Node		STRUCTURE
14	0E	dm_DimensionInfo	struct DimensionInfo	dm_DimensionInfo	STRUCT
102	66	dm_PropertyFlags	ULONG	dm_PropertyFlags	ULONG

DosEnvec

80 bytes
(Expanded for 2.0)

C Include File: dos/filehandler.h
ML Include File: dos/filehandler.i

byte	hex	name	C type	ML type
0	00	de_TableSize	ULONG	ULONG
4	04	de_SizeBlock	ULONG	ULONG
8	08	de_SecOrg	ULONG	ULONG
12	0C	de_Surfaces	ULONG	ULONG
16	10	de_SectorPerBlock	ULONG	ULONG
20	14	de_BlocksPerTrack	ULONG	ULONG
24	18	de_Reserved	ULONG	ULONG
28	1C	de_PreAlloc	ULONG	ULONG
32	20	de_Interleave	ULONG	ULONG
36	24	de_LowCyl	ULONG	ULONG
40	28	de_HighCyl	ULONG	ULONG
44	2C	de_NumBuffers	ULONG	ULONG
48	30	de_BufMemType	ULONG	ULONG
52	34	de_MaxTransfer	ULONG	ULONG
56	38	de_Mask	ULONG	ULONG
60	3C	de_BootPri	LONG	LONG
64	40	de_DosType	ULONG	ULONG
68	44	de_Baud	ULONG	ULONG
72	48	de_Control	ULONG	ULONG
76	4C	de_BootBlocks	ULONG	ULONG

Note: For de_DosType, $444F5300 is the old filesystem, $444F5301 is the fast filesystem.

DosInfo

158 bytes
(Updated for 3.0)

C Include File: dos/dosextens.h
ML Include File: dos/dosextens.i

byte	hex	name	C type	ML type
0	00	di_McName	BPTR	BPTR
4	04	di_DevInfo	BPTR	BPTR
8	08	di_Devices	BPTR	BPTR
12	0C	di_Handlers	BPTR	BPTR
16	10	di_NetHand	APTR	APTR
20	14	di_DevLock	struct SignalSemaphore	STRUCT
66	42	di_EntryLock	struct SignalSemaphore	STRUCT
112	70	di_DeleteLock	struct SignalSemaphore	STRUCT

DosLibrary

70 bytes
(Updated for 3.0)

C Include File: dos/dosextens.h
ML Include File: dos/dosextens.i

byte	hex	name	C type	ML type
0	00	dl_lib	struct Library	STRUCT
34	22	dl_Root	struct RootNode *	APTR
38	26	dl_GV	APTR	APTR
42	2A	dl_A2	LONG	LONG
46	2E	dl_A5	LONG	LONG
50	32	dl_A6	LONG	LONG
54	36	dl_Errors	struct ErrorString *	APTR
58	3A	dl_TimeReq	struct timerequest *	APTR
62	3E	dl_UtilityBase	struct Library *	APTR
66	42	dl_IntuitionBase	struct Library *	APTR

DosList

44 bytes

C Include File: dos/dosextens.h
ML Include File: dos/dosextens.i

byte	hex	name	C type	ML type
0	00	dol_Next	BPTR	BPTR
4	04	dol_Type	LONG	LONG
8	08	dol_Task	struct MsgPort *	APTR
12	0C	dol_Lock	BPTR	BPTR
16	10	dol_Handler	BSTR	BSTR

byte	hex	name	C type	ML type
20	14	dol_StackSize	LONG	LONG
24	18	dol_Priority	LONG	LONG
28	1C	dol_Startup	ULONG	ULONG
32	20	dol_SegList	BPTR	BPTR
36	24	dol_GlobVec	BPTR	BPTR
40	28	dol_Name	BSTR	BSTR

DosPacket

48 bytes
(Updated for 3.0)

C Include File: dos/dosextens.h
ML Include File: dos/dosextens.i

byte	hex	name	C type	ML type
0	00	dp_Link	struct Message *	APTR
4	04	dp_Port	struct MsgPort *	APTR
8	08	dp_Type	LONG	LONG
12	0C	dp_Res1	LONG	LONG
16	10	dp_Res2	LONG	LONG
20	14	dp_Arg1	LONG	LONG
24	18	dp_Arg2	LONG	LONG
28	1C	dp_Arg3	LONG	LONG
32	20	dp_Arg4	LONG	LONG
36	24	dp_Arg5	LONG	LONG
40	28	dp_Arg6	LONG	LONG
44	2C	dp_Arg7	LONG	LONG

Note: In machine language, the following equates exist: dp_Action EQU dp_Type, dp_Status EQU dp_Res1, dp_Status2 EQU dp_Res2, dp_BufAddr EQU dp_Arg1. This structure was previously located in libraries/dosextens.h (C version) and libraries/dosextens.i (ML version).

DrawerData

62 bytes
(Updated for 2.0)

C Include File: workbench/workbench.h
ML Include File: workbench/workbench.i

byte	hex	name	C type	ML type
0	00	dd_NewWindow	struct NewWindow	STRUCT
48	30	dd_CurrentX	LONG	LONG
52	34	dd_CurrentY	LONG	LONG
56	38	dd_Flags	ULONG	ULONG
60	3C	dd_ViewModes	UWORD	UWORD

DrawInfo

50 bytes
(Updated for 3.0)

C Include File: intuition/screens.h
ML Include File: intuition/screens.i

byte	hex	C name	C type	ML name	ML type
0	00	dri_Version	UWORD	dri_Version	UWORD
2	02	dri_NumPens	UWORD	dri_NumPens	UWORD
4	04	dri_Pens	UWORD *	dri_Pens	APTR
8	08	dri_Font	struct TextFont *	dri_Font	APTR
12	0C	dri_Depth	UWORD	dri_Depth	UWORD
14	0E	dri_Resolution.X	UWORD	dri_ResolutionX	UWORD
16	10	dri_Resolution.Y	UWORD	dri_ResolutionY	UWORD
18	12	dri_Flags	ULONG	dri_Flags	ULONG
22	16	dri_CheckMark	struct Image *	dri_CheckMark	APTR
26	1A	dri_AmigaKey	struct Image *	dri_AmigaKey	APTR
30	1E	dri_Reserved	ULONG[5]	sri_Reserved	STRUCT

dri_Flags

C flag name	ML flag name	ML bit name	bit	decimal	hex
DRIF_NEWLOOK	DRIF_NEWLOOK	DRIB_NEWLOOK	0	1	01

DriveGeometry

32 bytes
(New to 2.0)

C Include File: devices/trackdisk.h
ML Include File: devices/trackdisk.i

byte	hex	name	type
0	00	dg_SectorSize	ULONG
4	04	dg_TotalSectors	ULONG
8	08	dg_Cylinders	ULONG

byte	hex	name	type
12	0C	dg_CylSectors	ULONG
16	10	dg_Heads	ULONG
20	14	dg_TrackSectors	ULONG
24	18	dg_BufMemType	ULONG
28	1C	dg_DeviceType	UBYTE
29	1D	dg_Flags	UBYTE
30	1E	dg_Reserved	UWORD

dg_DeviceType

name	decimal	hex
DG_DIRECT_ACCESS	0	00
DG_SEQUENTIAL_ACCESS	1	01
DG_PRINTER	2	02
DG_PROCESSOR	3	03
DG_WORM	4	04
DG_CDROM	5	05
DG_SCANNER	6	06
DG_OPTICAL_DISK	7	07
DG_MEDIUM_CHANGER	8	08
DG_COMMUNICATION	9	09
DG_UNKNOWN	31	1F

dg_flags

C bit name	C flag name	ML bit name	bit	decimal	hex
DGB_REMOVABLE	DGF_REMOVABLE	DG,REMOVABLE	0	1	01

dtDraw

36 bytes
(New to 3.0)

C Include File: datatypes/datatypesclass.h
ML Include File: datatypes/datatypesclass.i

byte	hex	C name	C type	ML name	ML type
0	00	MethodID	ULONG		STRUCTURE
4	04	dtd_RPort	struct RastPort *	dtd_RPort	APTR
8	08	dtd_Left	LONG	dtd_Left	LONG
12	0C	dtd_Top	LONG	dtd_Top	LONG
16	10	dtd_Width	LONG	dtd_Width	LONG
20	14	dtd_Height	LONG	dtd_Height	LONG
24	18	dtd_TopHoriz	LONG	dtd_TopHoriz	LONG
28	1C	dtd_TopVert	LONG	dtd_TopVert	LONG
32	20	dtd_AttrList	struct TagItem *	dtd_AttrList	APTR

dtFrameBox

24 bytes
(New to 3.0)

C Include File: datatypes/datatypesclass.h
ML Include File: datatypes/datatypesclass.i

byte	hex	C name	C type	ML name	ML type
0	00	MethodID	ULONG		STRUCTURE
4	04	dtf_GInfo	struct GadgetInfo *	dtf_GInfo	APTR
8	08	dtf_ContentsInfo	struct FrameInfo *	dtf_ContentsInfo	APTR
12	0C	dtf_FrameInfo	struct FrameInfo *	dtf_FrameInfo	APTR
16	10	dtf_SizeFrameInfo	ULONG	dtf_SizeFrameInfo	ULONG
20	14	dtf_FrameFlags	ULONG	dtf_FrameFlags	ULONG

dtf_FrameFlags

C name	ML bit name	bit	decimal	hex
FRAMEF_SPECIFY	FRAMEF,SPECIFY	0	1	01

dtGeneral

8 bytes
(New to 3.0)

C Include File: datatypes/datatypesclass.h
ML Include File: datatypes/datatypesclass.i

byte	hex	C name	C type	ML name	ML type
0	00	MethodID	ULONG		STRUCTURE
4	04	dtg_GInfo	struct GadgetInfo *	dtg_GInfo	APTR

dtGoto

16 bytes
(New to 3.0)

C Include File: datatypes/datatypesclass.h
ML Include File: datatypes/datatypesclass.i

byte	hex	C name	C type	ML name	ML type
0	00	MethodID	ULONG		STRUCTURE
4	04	dtg_GInfo	struct GadgetInfo *	dtgo_GInfo	APTR
8	08	dtg_NodeName	STRPTR	dtgo_NodeName	APTR
12	0C	dtg_AttrList	struct TagItem *	dtgo_AttrList	APTR

DTHookContext

40 bytes
(New to 3.0)

C Include File: datatypes/datatypes.h
ML Include File: datatypes/datatypes.i

byte	hex	name	C type	ML type
0	00	dthc_SysBase	struct Library *	APTR
4	04	dthc_DOSBase	struct Library *	APTR
8	08	dthc_IFFParseBase	struct Library *	APTR
12	0C	dthc_UtilityBase	struct Library *	APTR
16	10	dthc_Lock	BPTR	BPTR
20	14	dthc_FIB	struct FileInfoBlock *	APTR
24	18	dthc_FileHandle	BPTR	BPTR
28	1C	dthc_IFF	struct IFFHandle *	APTR
32	20	dthc_Buffer	STRPTR	APTR
36	24	dthc_BufferLength	ULONG	ULONG

DTMethod

12 bytes
(New to 3.0)

C Include File: datatypes/datatypesclass.h
ML Include File: datatypes/datatypesclass.i

byte	hex	name	C type	ML type
0	00	dtm_Label	STRPTR	APTR
4	04	dtm_Command	STRPTR	APTR
8	08	dtm_Method	ULONG	ULONG

dtm_Method

C name	ML name	decimal	hex
DTM_Dummy		1536	0600
DTM_FRAMEBOX	DTM_FRAMEBOX	1537	0601
DTM_PROCLAYOUT	DTM_PROCLAYOUT	1538	0602

C name	ML name	decimal	hex	DTM_Dummy
DTM_ASYNCLAYOUT	DTM_ASYNCLAYOUT	1539	0603	
DTM_REMOVEDOBJECT	DTM_REMOVEDOBJECT	1540	0604	
DTM_SELECT	DTM_SELECT	1541	0605	
DTM_CLEARSELECTED	DTM_CLEARSELECTED	1542	0606	
DTM_COPY	DTM_COPY	1543	0607	
DTM_PRINT	DTM_PRINT	1544	0608	
DTM_ABORTPRINT	DTM_ABORTPRINT	1545	0609	
DTM_NEWMEMBER	DTM_NEWMEMBER	1546	060A	
DTM_DISPOSEMEMBER	DTM_DISPOSEMEMBER	1547	060B	
DTM_GOTO	DTM_GOTO	1584	0630	
DTM_TRIGGER	DTM_TRIGGER	1685	0631	
DTM_OBTAINDRAWINFO	DTM_OBTAINDRAWINFO	1600	0640	
DTM_DRAW	DTM_DRAW	1601	0641	
DTM_RELEASEDRAWINFO	DTM_RELEASEDRAWINFO	1602	0642	
DTM_WRITE	DTM_WRITE	1616	0650	

dtSelect

16 bytes
(New to 3.0)

C Include File: datatypes/datatypesclass.h
ML Include File: datatypes/datatypesclass.i

byte	hex	C name	C type	ML name	ML type
0	00	MethodID	ULONG		STRUCTURE
4	04	dts_GInfo	struct GadgetInfo *	dts_GInfo	APTR
8	08	dts_Select	struct Rectangle	dts_Select	STRUCT

DTSpecialInfo

90 bytes
(New to 3.0)

C Include File: datatypes/datatypesclass.h
ML Include File: datatypes/datatypesclass.i

byte	hex	name	C type	ML type
0	00	si_Lock	struct SignalSemaphore	STRUCT
46	2E	si_Flags	ULONG	ULONG
50	32	si_TopVert	LONG	LONG
54	36	si_VisVert	LONG	LONG
58	3A	si_TotVert	LONG	LONG
62	3E	si_OTopVert	LONG	LONG

byte	hex	name	C type	ML type
66	42	si_VertUnit	LONG	LONG
70	46	si_TopHoriz	LONG	LONG
74	4A	si_VisHoriz	LONG	LONG
78	4E	si_TotHoriz	LONG	LONG
82	52	si_OTopHoriz	LONG	LONG
86	56	si_HorizUnit	LONG	LONG

si_Flags

C bit name	ML bit name	bit	decimal	hex
DTSIF_LAYOUT	DTSI,LAYOUT	0	1	01
DTSIF_NEWSIZE	DTSI,NEWSIZE	1	2	02
DTSIF_DRAGGING	DTSI,DRAGGING	2	4	04
DTSIF_DRAGSELECT	DTSI,DRAGSELECT	3	8	08
DTSIF_HIGHLIGHT	DTSI,HIGHLIGHT	4	16	10
DTSIF_PRINTING	DTSI,PRINTING	5	32	20
DTSIF_LAYOUTPROC	DTSI,LAYOUTPROC	6	64	40

dtTrigger

16 bytes
(New to 3.0)

C Include File: datatypes/datatypesclass.h
ML Include File: datatypes/datatypesclass.i

byte	hex	C name	C type	ML name	ML type
0	00	MethodID	ULONG		STRUCTURE
4	04	dtt_GInfo	struct GadgetInfo *	dtt_GInfo	APTR
8	08	dtt_Function	ULONG	dtt_Function	ULONG
12	0C	dtt_Data	APTR	dtt_Data	APTR

dtt_Function

name	decimal	hex
STM_PAUSE	1	01
STM_PLAY	2	02
STM_CONTENTS	3	03
STM_INDEX	4	04
STM_RETRACE	5	05
STM_BROWSE_PREV	6	06
STM_BROWSE_NEXT	7	07
STM_NEXT_FIELD	8	08
STM_PREV_FIELD	9	09
STM_ACTIVATE_FIELD	10	0A
STM_COMMAND	11	0B

dtWrite

20 bytes
(New to 3.0)

C Include File: datatypes/datatypesclass.h
ML Include File: datatypes/datatypesclass.i

byte	hex	C name	C type	ML name	ML type
0	00	MethodID	ULONG		STRUCTURE
4	04	dtw_GInfo	struct GadgetInfo *	dtw_GInfo	APTR
8	08	dtw_FileHandle	BPTR	dtw_FileHandle	BPTR
12	0C	dtw_Mode	ULONG	dtw_Mode	ULONG
16	10	dtw_AttrList	struct TagItem *	dtw_AttrList	APTR

dtw_Mode

name	decimal	hex
DTWM_IFF	0	00
DTWM_RAW	1	01

EasyStruct

20 bytes
(New to 2.0)

C Include File: intuition/intuition.h
ML Include File: intuition/intuition.i

byte	hex	C name	C type	ML type
0	00	es_StructSize	ULONG	ULONG
4	04	es_Flags	ULONG	ULONG
8	08	es_Title	UBYTE *	APTR
12	0C	es_TextFormat	UBYTE *	APTR
16	10	es_GadgetFormat	UBYTE *	APTR

Note: es_Flags is currently 0.

EClockVal

(ECLOCKVAL in ML)
8 bytes
(New to 2.0)

C Include File: devices/timer.h
ML Include File: devices/timer.i

byte	hex	C name	C type	ML name	ML type
0	00	ev_hi	ULONG	EV_HI	ULONG
4	04	ev_lo	ULONG	EV_LO	ULONG

ErrorString

8 bytes
(New to 3.0)

C Include File: dos/dosextens.h
ML Include File: dos/dosextens.i

byte	hex	name	C type	ML type
0	00	estr_Nums	LONG *	APTR
4	04	estr_Strings	UBYTE *	APTR

ExAllControl

16 bytes
(New to 2.0)

C Include File: dos/exall.h
ML Include File: dos/exall.i

byte	hex	name	C type	ML type
0	00	eac_Entries	ULONG	ULONG
4	04	eac_LastKey	ULONG	ULONG
8	08	eac_MatchString	UBYTE *	APTR
12	0C	eac_MatchFunc	struct Hook *	APTR

ExAllData

40 bytes
(Updated for 3.0)

C Include File: dos/exall.h
ML Include File: dos/exall.i

byte	hex	name	C type	ML type
0	00	ed_Next	struct ExAllData *	APTR
4	04	ed_Name	UBYTE *	APTR
8	08	ed_Type	LONG	LONG
12	0C	ed_Size	ULONG	ULONG
16	10	ed_Prot	ULONG	ULONG
20	14	ed_Days	ULONG	ULONG
24	18	ed_Mins	ULONG	ULONG
28	1C	ed_Ticks	ULONG	ULONG
32	20	ed_Comment	UBYTE *	APTR
36	24	ed_OwnerUID	UWORD	UWORD
38	26	ed_OwnerGID	UWORD	UWORD

ExecBase

632 bytes
(Updated for 3.0)

C Include File: exec/execbase.h
ML Include File: exec/execbase.i

byte	hex	name	C type	ML type
0	0000	LibNode	struct Library	STRUCTURE
34	0022	SoftVer	UWORD	UWORD
36	0024	LowMemChkSum	WORD	WORD
38	0026	Chkbase	ULONG	ULONG
42	002A	ColdCapture	APTR	APTR
46	002E	CoolCapture	APTR	APTR
50	0032	WarmCapture	APTR	APTR
54	0036	SysStkUpper	APTR	APTR
58	003A	SysStkLower	APTR	APTR
62	003E	MaxLocMem	ULONG	ULONG
66	0042	DebugEntry	APTR	APTR
70	0046	DebugData	APTR	APTR
74	004A	AlertData	APTR	APTR
78	004E	MaxExtMem	APTR	APTR
82	0052	ChkSum	UWORD	WORD

byte	hex	name	C type	ML type
84	0054	IntVects	struct IntVector[16]	STRUCT
276	0114	ThisTask	struct Task *	APTR
280	0118	IdleCount	ULONG	ULONG
284	011C	DispCount	ULONG	ULONG
288	0120	Quantum	UWORD	UWORD
290	0122	Elapsed	UWORD	UWORD
292	0124	SysFlags	UWORD	UWORD
294	0126	IDNestCnt	BYTE	BYTE
295	0127	TDNestCnt	BYTE	BYTE
296	0128	AttnFlags	UWORD	UWORD
298	012A	AttnResched	UWORD	UWORD
300	012C	ResModules	APTR	APTR
304	0130	TaskTrapCode	APTR	APTR
308	0134	TaskExceptCode	APTR	APTR
312	0138	TaskExitCode	APTR	APTR
316	013C	TaskSignalAlloc	ULONG	ULONG
320	0140	TaskTrapAlloc	UWORD	UWORD
322	0142	MemList	struct List	STRUCT
336	0150	ResourceList	struct List	STRUCT
350	015E	DeviceList	struct List	STRUCT
364	016C	IntrList	struct List	STRUCT
378	017A	LibList	struct List	STRUCT
392	0188	PortList	struct List	STRUCT
406	0196	TaskReady	struct List	STRUCT
420	01A4	TaskWait	struct List	STRUCT
434	01B2	SoftInts	struct SoftIntList[5]	STRUCT
514	0202	LastAlert	LONG[4]	STRUCT
530	0212	VBlankFrequency	UBYTE	UBYTE
531	0213	PowerSupplyFrequency	BYTE	UBYTE
532	0214	SemaphoreList	struct List	STRUCT
546	0222	KickMemPtr	APTR	APTR
550	0226	KickTagPtr	APTR	APTR
554	022A	KickCheckSum	APTR	APTR
558	022E	ex_Pad0	UWORD	UWORD
560	0230	ex_LaunchPoint	ULONG	ULONG
564	0234	ex_RamLibPrivate	APTR	APTR
568	0238	ex_EClockFrequency	ULONG	ULONG
572	023C	ex_CacheControl	ULONG	ULONG
576	0240	ex_TaskID	ULONG	ULONG
580	0244	ex_Reserved1	ULONG[5]	STRUCT
600	0258	ex_MMULock	APTR	APTR
604	025C	ex_Reserved2	ULONG[3]	STRUCT
616	0268	ex_MemHandlers	struct MinList	STRUCT
628	0274	ex_MemHandler	APTR	APTR

AttnFlags

C flag name	C bit name	ML bit name	bit	decimal	hex
AFF_68010	AFB_68010	AF,68010	0	1	01
AFF_68020	AFB_68020	AF,68020	1	2	02
AFF_68030	AFB_68030	AF,68030	2	4	04
AFF_68040	AFB_68040	AF,68040	3	8	08
AFF_68881	AFB_68881	AF,68881	4	16	10
AFF_68882	AFB_68882	AF,68882	5	32	20
AFF_FPU40	AFB_FPU40	AF,FPU40	6	64	40

ExpansionBase

88 bytes
(Updated for 3.0)

C Include File: libraries/expansionbase.h
ML Include File: libraries/expansionbase.i

byte	hex	C name	C type	ML name	ML type
0	0000	LibNode	struct Library	eb_LibNode	STRUCTURE
34	0022	Flags	UBYTE	eb_Flags	UBYTE
35	0023	eb_Private01	UBYTE	eb_Private01	UBYTE
36	0024	eb_Private02	ULONG	eb_Private02	ULONG
40	0028	eb_Private03	ULONG	eb_Private03	ULONG
44	002C	eb_Private04	struct CurrentBinding	eb_Private04	STRUCT
60	003C	eb_Private05	struct List	eb_Private05	STRUCT
74	004A	MountList	struct List	eb_MountList	STRUCT

Note: This structure has been redefined in 2.0 and 3.0 to be more private. This structure is not supposed to be accessed by any user code.

Flags (eb_Flags in ML)

C flag name	C bit name	ML bit name	bit	decimal	hex
EBF_CLOGGED	EBB_CLOGGED	EB,CLOGGED	0	1	01
EBF_SHORTMEM	EBB_SHORTMEM	EB,SHORTMEM	1	2	02
EBF_BADMEM	EBB_BADMEM	EB,BADMEM	2	4	04
EBF_DOSFLAG	EBB_DOSFLAG	EB,DOSFLAG	3	8	08
EBF_KICKBACK33	EBB_KICKBACK33	EB,KICKBACK33	4	16	10
EBF_KICKBACK36	EBB_KICKBACK36	EB,KICKBACK36	5	32	20
EBF_SILENTSTART	EBB_SILENTSTART	EB,SILENTSTART	6	64	40
EBF_START_CC0	EBB_START_CC0	EB,START_CC0	7	128	80

ExpansionControl

16 bytes
(Updated for 3.0)

C Include File: libraries/configregs.h
ML Include File: libraries/configregs.i

byte	hex	name	type
0	00	ec_Interrupt	UBYTE
1	01	ec_Z3_HighBase	UBYTE
2	02	ec_BaseAddress	UBYTE
3	03	ec_ShutUp	UBYTE
4	04	ec_Reserved14	UBYTE
5	05	ec_Reserved15	UBYTE
6	06	ec_Reserved16	UBYTE
7	07	ec_Reserved17	UBYTE
8	08	ec_Reserved18	UBYTE
9	09	ec_Reserved19	UBYTE
10	0A	ec_Reserved1a	UBYTE
11	0B	ec_Reserved1b	UBYTE
12	0C	ec_Reserved1c	UBYTE
13	0D	ec_Reserved1d	UBYTE
14	0E	ec_Reserved1e	UBYTE
15	0F	ec_Reserved1f	UBYTE

ExpansionRom

16 bytes

C Include File: libraries/configregs.h
ML Include File: libraries/configregs.i

byte	hex	name	type
0	00	er_Type	UBYTE
1	01	er_Product	UBYTE
2	02	er_Flags	UBYTE
3	03	er_Reserved03	UBYTE
4	04	er_Manufacturer	UWORD
6	06	er_SerialNumber	ULONG
10	0A	er_InitDiagVec	UWORD
12	0C	er_Reserved0c	UBYTE

byte	hex	name	type
13	0D	er_Reserved0d	UBYTE
14	0E	er_Reserved0e	UBYTE
15	0F	er_Reserved0f	UBYTE

Note: This structure was previously found in libraries/configvars.h (C version) and libraries/configvars.i (ML version).

ExtendedNode (XLN in ML)

24 bytes
(New to 2.0)

C Include File: graphics/gfxnodes.h
ML Include File: graphics/gfxnodes.i

byte	hex	C name	C type	ML name	ML type
0	00	xln_Succ	struct Node *	XLN_SUCC	APTR
4	04	xln_Pred	struct Node *	XLN_PRED	APTR
8	08	xln_Type	UBYTE	XLN_TYPE	UBYTE
9	09	xln_Pri	BYTE	XLN_PRI	BYTE
10	0A	xln_Name	char *	XLN_NAME	APTR
14	0E	xln_Subsystem	UBYTE	XLN_SUBSYSTEM	UBYTE
15	0F	xln_Subtype	UBYTE	XLN_SUBTYPE	UBYTE
16	10	xln_Library	LONG	XLN_LIBRARY	LONG
20	14	xln_Init	LONG(*)()	XLN_INIT	LONG

xln_Type (XLN_TYPE in ML)

name	decimal	hex
VIEW_EXTRA_TYPE	1	01
VIEWPORT_EXTRA_TYPE	2	02
SPECIAL_MONITOR_TYPE	3	03
MONITOR_SPEC_TYPE	4	04

ExtGadget

56 bytes
(New to 3.0)

C Include File: intuition/intuition.h
ML Include File: intuition/intuition.i

byte	hex	C name	C type	ML Name	ML type
0	00	NextGadget	struct ExtGadget *	egg_NextGadget	APTR
4	04	LeftEdge	WORD	egg_LeftEdge	WORD
6	06	TopEdge	WORD	egg_TopEdge	WORD
8	08	Width	WORD	egg_Width	WORD
10	0A	Height	WORD	egg_Height	WORD
12	0C	Flags	UWORD	egg_Height	WORD
14	0E	Activation	UWORD	egg_Activation	WORD
16	10	GadgetType	UWORD	egg_GadgetType	WORD
18	12	GadgetRender	APTR	egg_GadgetRender	APTR
22	16	SelectRender	APTR	egg_SelectRender	APTR
26	1A	GadgetText	struct IntuiText	egg_GadgetText	APTR
30	1E	MutualExclude	LONG	egg_MutualExclude	LONG
34	22	SpecialInfo	APTR	egg_SpecialInfo	APTR
38	26	GadgetID	UWORD	egg_GadgetID	WORD
40	28	UserData	APTR	egg_UserData	APTR
44	2C	MoreFlags	ULONG	egg_MoreFlags	ULONG
48	30	BoundsLeftEdge	WORD	egg_BoundsLeftEdge	WORD
50	32	BoundsTopEdge	WORD	egg_BoundsTopEdge	WORD
52	34	BoundsWidth	WORD	egg_BoundsWidth	WORD
54	36	BoundsHeight	WORD	egg_BoundsHeight	WORD

Flags

name	bit	decimal	hex
GADGIMAGE	2	4	0004
GRELBOTTOM	3	8	0008
GRELRIGHT	4	16	0010
GRELWIDTH	5	32	0020
name	**bit**	**decimal**	**hex**
GRELHEIGHT	6	64	0040
SELECTED	7	128	0080
GADGDISABLED	8	256	0100

Activation

name	bit	decimal	hex
RELVERIFY	0	1	0001
GADGIMMEDIATE	1	2	0002
ENDGADGET	2	4	0004
FOLLOWMOUSE	3	8	0008
RIGHTBORDER	4	16	0010
LEFTBORDER	5	32	0020
TOPBORDER	6	64	0040
BOTTOMBORDER	7	128	0080

TOGGLESELECT	8	256	0100
STRINGCENTER	9	512	0200
STRINGRIGHT	10	1024	0400
LONGINT	11	2048	0800
ALTKEYMAP	12	4096	1000
BOOLEXTEND	13	8192	2000

ExtIntuiMessage

56 bytes
(New to 3.0)

C Include File: intuition/intuition.h
ML Include File: intuition/intuition.i

byte	hex	name	C type	ML type
0	00	eim_IntuiMessage	struct IntuiMessage	STRUCT
52	34	eim_TabletData	struct TabletData *	APTR

ExtNewScreen

36 bytes
(New to 2.0)

C Include File: intuition/screens.h
ML Include File: intuition/screens.i

byte	hex	C name	C type	ML name	ML type
0	00				STRUCTURE
0	00	LeftEdge	WORD		
2	02	TopEdge	WORD		
4	04	Width	WORD		
6	06	Height	WORD		
8	08	Depth	WORD		
10	0A	DetailPen	UBYTE		
11	0B	BlockPen	UBYTE		
12	0C	ViewModes	UWORD		
14	0E	Type	UWORD		
16	10	Font	struct TextAttr *		
20	14	DefaultTitle	UBYTE *		
24	18	Gadgets	struct Gadget *		
28	1C	CustomBitMap	struct BitMap *		
32	20	Extension	struct TagItem *	ens_Extension	APTR

ViewModes

C name	ML name	bit	decimal	hex
GENLOCK_VIDEO	GENLOCK VIDEO	1	2	0002
LACE	V_LACE	2	4	0004
PFBA	V_PFBA	6	64	0040
EXTRA_HALFBRITE		7	128	0080
GENLOCK_AUDIO		8	256	0100
DUALPF	V_DUALPF	10	1024	0400
HAM	V_HAM	11	2048	0800
VP_HIDE		13	8192	2000
SPRITES	V_SPRITES	14	16384	4000
HIRES	V_HIRES	15	32768	8000

Type

name	bit	decimal	hex
SHOWTITLE	4	16	0010
BEEPING	5	32	0020
CUSTOMBITMAP	6	64	0040
SCREENBEHIND	7	128	0080
SCREENQUIET	8	256	0100

ExtNewWindow

52 bytes
(New to 2.0)

C Include File: intuition/intuition.h
ML Include File: intuition/intuition.i

byte	hex	C name	C type	ML name	ML type
0	00				STRUCTURE
0	00	LeftEdge	WORD		
2	02	TopEdge	WORD		
4	04	Width	WORD		
6	06	Height	WORD		
8	08	DetailPen	UBYTE		
9	09	BlockPen	UBYTE		
10	0A	IDCMPFlags	ULONG		
14	0E	Flags	ULONG		
18	12	FirstGadget	struct Gadget *		
22	16	CheckMark	struct Image *		
26	1A	Title	UBYTE *		
30	1E	Screen	struct Screen *		

byte	hex	C name	C type	ML name	ML type
34	22	BitMap	struct BitMap *		
38	26	MinWidth	WORD		
40	28	MinHeight	WORD		
42	2A	MaxWidth	UWORD		
44	2C	MaxHeight	UWORD		
46	2E	Type	UWORD		
48	30	Extension	struct TagItem *	enw_Extension	APTR

IDCMPFlags

name	bit	decimal	hex
SIZEVERIFY	0	1	00000001
NEWSIZE	1	2	00000002
REFRESHWINDOW	2	4	00000004
MOUSEBUTTONS	3	8	00000008
MOUSEMOVE	4	16	00000010
GADGETDOWN	5	32	00000020
GADGETUP	6	64	00000040
REQSET	7	128	00000080
MENUPICK	8	256	00000100
CLOSEWINDOW	9	512	00000200
RAWKEY	10	1024	00000400
REQVERIFY	11	2048	00000800
REQCLEAR	12	4096	00001000
MENUVERIFY	13	8192	00002000
NEWPREFS	14	16384	00004000
DISKINSERTED	15	32768	00008000
DISKREMOVED	16	65536	00010000
WBENCHMESSAGE	17	131072	00020000
ACTIVEWINDOW	18	262144	00040000
INACTIVEWINDOW	19	524288	00080000
DELTAMOVE	20	1048576	00100000
VANILLAKEY	21	2097152	00200000
INTUITICKS	22	4194304	00400000
LONELYMESSAGE	31	2147483648	80000000

Flags

name	bit	decimal	hex
WINDOWSIZING	0	1	00000001
WINDOWDRAG	1	2	00000002
WINDOWDEPTH	2	4	00000004
WINDOWCLOSE	3	8	00000008
SIZEBRIGHT	4	16	00000010
SIZEBBOTTOM	5	32	00000020
BACKDROP	8	256	00000100

name	bit	decimal	hex
REPORTMOUSE	9	512	00000200
GIMMEZEROZERO	10	1024	00000400
BORDERLESS	11	2048	00000800
ACTIVATE	12	4096	00001000
WINDOWACTIVE	13	8192	00002000
INREQUEST	14	16384	00004000
MENUSTATE	15	32768	00008000
RMBTRAP	16	65536	00010000
NOCAREREFRESH	17	131072	00020000
WINDOWREFRESH	24	16777216	01000000
WBENCHWINDOW	25	33554432	02000000
WINDOWTICKED	26	67108864	04000000

Type

name	bit	decimal	hex
SHOWTITLE	4	16	0010
BEEPING	5	32	0020
CUSTOMBITMAP	6	64	0040
SCREENBEHIND	7	128	0080
SCREENQUIET	8	256	0100

ExtSprite

16 bytes
(New to 3.0)

C Include File: graphics/sprite.h
ML Include File: graphics/sprite.i

byte	hex	name	C type	ML type
0	00	es_SimpleSprite	struct SimpleSprite	STRUCT
12	0C	es_wordwidth	UWORD	WORD
14	0E	es_flags	UWORD	WORD

FC

see FontContents

FCH

see FontContentsHeader

FileHandle

44 bytes
(Updated for 3.0)

C Include File: dos/dosextens.h
ML Include File: dos/dosextens.i

byte	hex	C name	C type	ML Name	ML type
0	00	fh_Link	struct Message *	fh_Link	APTR
4	04	fh_Port	struct MsgPort *	fh_Interactive	APTR
8	08	fh_Type	struct MsgPort *	fh_Type	APTR
12	0C	fh_Buf	LONG	fh_Buf	LONG
16	10	fh_Pos	LONG	fh_Pos	LONG
20	14	fh_End	LONG	fh_End	LONG
24	18	fh_Func1	LONG	fh_Func1	LONG
28	1C	fh_Func2	LONG	fh_Func2	LONG
32	20	fh_Func3	LONG	fh_Func3	LONG
36	24	fh_Arg1	LONG	fh_Arg1	LONG
40	48	fh_Arg2	LONG	fh_Arg2	LONG

Note: The elements fh_Func1 and fh_Arg1 may also be referred to as fh_Funcs and fh_Args, respectively. This structure was previously found in libraries/dosextens.h (C version) and libraries/dosextens.i (ML version).

FileInfoBlock

260 bytes
(Updated for 3.0)

C Include File: dos/dos.h
ML Include File: dos/dos.i

byte	hex	name	C type	ML type
0	00	fib_DiskKey	LONG	LONG
4	04	fib_DirEntryType	LONG	LONG
8	08	fib_FileName	char[108]	STRUCT
116	74	fib_Protection	LONG	LONG
120	78	fib_EntryType	LONG	LONG
124	7C	fib_Size	LONG	LONG
128	80	fib_NumBlocks	LONG	LONG
132	84	fib_Date	struct DateStamp	STRUCT
144	90	fib_Comment	char[80]	STRUCT

byte	hex	name	C type	ML type
224	E0	fib_OwnerUID	UWORD	UWORD
226	E2	fib_OwnerGID	UWORD	UWORD
228	E4	fib_Reserved	char[32]	STRUCT

fib_Protection

C flag name	C bit name	ML bit name	bit	decimal	hex
FIBF_DELETE	FIBB_DELETE	FIB,DELETE	0	1	01
FIBF_EXECUTE	FIBB_EXECUTE	FIB,EXECUTE	1	2	02
FIBF_WRITE	FIBB_WRITE	FIB,WRITE	2	4	04
FIBF_READ	FIBB_READ	FIB,READ	3	8	08
FIBF_ARCHIVE	FIBB_ARCHIVE	FIB,ARCHIVE	4	16	10
FIBF_PURE	FIBB_PURE	FIB,PURE	5	32	20
FIBF_SCRIPT	FIBB_SCRIPT	FIB,SCRIPT	6	64	40

Note: This structure was previously found in libraries/dos.h (C version) and libraries/dos.i (ML version).

FileLock

20 bytes

C Include File: dos/dosextens.h
ML Include File: dos/dosextens.i

byte	hex	name	C type	ML type
0	00	fl_Link	BPTR	BPTR
4	04	fl_Key	LONG	LONG
8	08	fl_Access	LONG	LONG
12	0C	fl_Task	struct MsgPort *	APTR
16	10	fl_Volume	BPTR	BPTR

Note: This structure was previously found in libraries/dosextens.h (C version) and libraries/dosextens.i (ML version).

FileRequester

56 bytes
(Updated for 3.0)

C Include File: libraries/asl.h
ML Include File: libraries/asl.i

byte	hex	name	C type	ML type
0	00	rf_Reserved0	UBYTE[4]	STRUCTURE
4	04	rf_File	STRPTR	APTR
8	08	rf_Drawer	STRPTR	APTR
12	0C	rf_Reserved1	UBYTE[10]	STRUCT
22	16	rf_LeftEdge	WORD	WORD
24	18	rf_TopEdge	WORD	WORD
26	1A	rf_Width	WORD	WORD
28	1C	rf_Height	WORD	WORD
30	1E	rf_Reserved2	UBYTE[2]	STRUCT
32	20	rf_NumArgs	LONG	LONG
36	24	rf_ArgList	struct WBArg *	APTR
40	28	rf_UserData	APTR	APTR
44	2C	rf_Reserved3	APTR	STRUCT
52	34	rf_Pattern	STRPTR	APTR

FileSysEntry

62 bytes
(New to 2.0)

C Include File: resources/filesysres.h
ML Include File: resources/filesysres.i

byte	hex	name	C type	ML type
0	00	fse_Node	struct Node	STRUCTURE
14	0E	fse_DosType	ULONG	ULONG
18	12	fse_Version	ULONG	ULONG
22	16	fse_PatchFlags	ULONG	ULONG
26	1A	fse_Type	ULONG	ULONG
30	1E	fse_Task	CPTR	CPTR
34	22	fse_Lock	BPTR	BPTR
38	26	fse_Handler	BSTR	BSTR
42	2A	fse_StackSize	ULONG	ULONG
46	2E	fse_Priority	LONG	LONG
50	32	fse_Startup	BPTR	BPTR
54	36	fse_SegList	BPTR	BPTR
58	3A	fse_GlobalVec	BPTR	BPTR

fse_PatchFlags

substitution	bit	decimal	hex
(fse_Type)	0	1	0001
(fse_Task)	1	2	0002
(fse_Lock)	2	4	0004

substitution	bit	decimal	hex
(fse_Handler)	3	8	0008
(fse_StackSize)	4	16	0010
(fse_Priority)	5	32	0020
(fse_Startup)	6	64	0040
(fse_SegList)	7	128	0080
(fse_GlobalVec)	8	256	0100

Note: For every entry that you wish to copy from this structure to the device node, set the appropriate bit in fse_PatchFlags. There are no names assigned to these bits in C or ML.

FileSysHeaderBlock

256 bytes
(New to 2.0)

C Include File: devices/hardblocks.h
ML Include File: devices/hardblocks.i

byte	hex	name	C type	ML type
0	00	fhb_ID	ULONG	ULONG
4	04	fhb_SummedLongs	ULONG	ULONG
8	08	fhb_ChkSum	LONG	LONG
12	0C	fhb_HostID	ULONG	ULONG
16	10	fhb_Next	ULONG	ULONG
20	14	fhb_Flags	ULONG	ULONG
24	18	fhb_Reserved1	ULONG[2]	STRUCT
32	20	fhb_DosType	ULONG	ULONG
36	24	fhb_Version	ULONG	ULONG
40	28	fhb_PatchFlags	ULONG	ULONG
44	2C	fhb_Type	ULONG	ULONG
48	30	fhb_Task	ULONG	ULONG
52	34	fhb_Lock	ULONG	ULONG
56	38	fhb_Handler	ULONG	ULONG
60	3C	fhb_StackSize	ULONG	ULONG
64	40	fhb_Priority	LONG	LONG
68	44	fhb_Startup	LONG	LONG
72	48	fhb_SegListBlocks	LONG	LONG
76	4C	fhb_GlobalVec	LONG	LONG
80	50	fhb_Reserved2	ULONG[23]	STRUCT
172	AC	fhb_Reserved3	ULONG[21]	STRUCT

fhb_PatchFlags

substitution	bit	decimal	hex
(fhb_Type)	0	1	0001
(fhb_Task)	1	2	0002
(fhb_Lock)	2	4	0004
(fhb_Handler)	3	8	0008
(fhb_StackSize)	4	16	0010
(fhb_Priority)	5	32	0020
(fhb_Startup)	6	64	0040
(fhb_SegList)	7	128	0080
(fhb_GlobalVec)	8	256	0100

FileSysResource

32 bytes
(New to 2.0)

C Include File: resources/filesysres.h
ML Include File: resources/filesysres.i

byte	hex	name	C type	ML type
0	00			STRUCTURE
0	00	fsr_Node	struct Node	
4	04	fsr_Creator	char *	CPTR
8	08	fsr_FileSysEntries	struct List	STRUCT

FileSysStartupMsg

16 bytes

C Include File: dos/filehandler.h
ML Include File: dos/filehandler.i

byte	hex	name	C type	ML type
0	00	fssm_Unit	ULONG	ULONG
4	04	fssm_Device	BSTR	BSTR
8	08	fssm_Environ	BPTR	BPTR
12	0C	fssm_Flags	ULONG	ULONG

FontContents

(FC in ML)
260 bytes

C Include File: diskfont/diskfont.h
ML Include File: diskfont/diskfont.i

byte	hex	name	C type	ML type
0	0000	fc_FileName	char[256]	STRUCT
256	0100	fc_YSize	UWORD	UWORD
258	0102	fc_Style	UBYTE	UBYTE
259	0103	fc_Flags	UBYTE	UBYTE

fc_Style

C flag name	C bit name	ML name	bit	decimal	hex
FSF_UNDERLINED	FSB_UNDERLINED	FS,UNDERLINED	0	1	01
FSF_BOLD	FSB_BOLD	FS,BOLD	1	2	02
FSF_ITALIC	FSB_ITALIC	FS,ITALIC	2	4	04
FSF_EXTENDED	FSB_EXTENDED	FS,EXTENDED	3	8	08

fc_Flags

C flag name	C bit name	ML name	bit	decimal	hex
FPF_ROMFONT	FPB_ROMFONT	FP,ROMFONT	0	1	01
FPF_DISKFONT	FPB_DISKFONT	FP,DISKFONT	1	2	02
FPF_REVPATH	FPB_REVPATH	FP,REVPATH	2	4	04
FPF_TALLDOT	FPB_TALLDOT	FP,TALLDOT	3	8	08
FPF_LONGDOT	FPB_LONGDOT	FP,LONGDOT	4	16	10
FPF_PROPORTIONAL	FPB_PROPORTIONAL	FP,PROPORTIONAL	5	32	20
FPF_DESIGNED	FPB_DESIGNED	FP,DESIGNED	6	64	40
FPF_REMOVED	FPB_REMOVED	FP,REMOVED	7	128	80

Note: This structure was previously found in libraries/diskfont.h (C version) and libraries/diskfont.i (ML version).

FontContentsHeader

4 bytes

C Include File: diskfont/diskfont.h
ML Include File: diskfont/diskfont.i

byte	hex	name	C type	ML type
0	00	fch_FileID	UWORD	UWORD
2	02	fch_NumEntries	UWORD	UWORD

Note: This structure was previously found in libraries/diskfont.h (C version) and libraries/diskfont.i (ML version).

FontPrefs

156 bytes
(New to 3.0)

C Include File: prefs/font.h
ML Include File: prefs/font.i

byte	hex	name	C type	ML type
0	00	fp_Reserved	LONG[3]	STRUCT
12	0C	fp_Reserved2	UWORD	UWORD
14	0E	fp_Type	UWORD	UWORD
16	10	fp_FrontPen	UBYTE	UBYTE
byte	**hex**	**name**	**C type**	**ML type**
17	11	fp_BackPen	UBYTE	UBYTE
18	12	fp_DrawMode	UBYTE	UBYTE
20	14	fp_TextAttr	struct TextAttr	STRUCT
28	1C	fp_Name	BYTE[FONTNAMESIZE]	STRUCT

fp_Type

name	decimal	hex
FP_WBFONT	0	00
FP_SYSFONT	1	01
FP_SCREENFONT	2	02

FontRequester

24 bytes
(Updated for 3.0)

C Include File: libraries/asl.h
ML Include File: libraries/asl.i

byte	hex	C name	C type	ML name	ML type
0	00	fo_Reserved0	UBYTE[8]		
0	00			FontRequester	STRUCTURE
4	04	fo_Attr	struct TextAttr	fo_Attr	STRUCT
12	0C	fo_FrontPen	UBYTE	fo_FrontPen	UBYTE
13	0D	fo_BackPen	UBYTE	fo_BackPen	UBYTE
14	0E	fo_DrawMode	UBYTE	fo_DrawMode	UBYTE
15	0F	fo_Reserved1	UBYTE	fo_Reserved1	UBYTE
16	10	fo_UserData	APTR	fo_UserData	APTR
20	14	fo_LeftEdge	WORD	fo_LeftEdge	WORD
22	16	fo_TopEdge	WORD	fo_TopEdge	WORD
24	18	fo_Width	WORD	fo_Width	WORD
26	1A	fo_Height	WORD	fo_Height	WORD
28	1C	fo_TAttr	struct TTextAttr	fo_TAttr	STRUCT

FrameInfo

36 bytes
(New to 3.0)

C Include File: datatypes/datatypesclass.h
ML Include File: datatypes/datatypesclass.i

byte	hex	C name	C type	ML name	ML type
0	00	fri_PropertyFlags	ULONG	fri_PropertyFlags	ULONG
4	04	fri_Resolution	Point	fri_Resolution	STRUCT
8	08	fri_RedBits	UBYTE	fri_RedBits	UBYTE
9	09	fri_GreenBits	UBYTE	fri_GreenBits	UBYTE
10	0A	fri_BlueBits	UBYTE	fri_GreenBits	UBYTE
12	0C	fri_Dimensions	struct{ULONG Width; ULONG Height; ULONG Depth;}		
12	0C			fri_Width	ULONG
16	10			fri_Height	ULONG
20	14			fri_Depth	ULONG
24	18	fri_Screen	Screen *	fri_Screen	APTR
28	1C	fri_ColorMap	ColorMap *	fri_ColorMap	APTR
32	20	fri_Flags	ULONG	fri_Flags	ULONG

fri_Flags

C name	ML bit name	bit	decimal	hex
FIF_SCALABLE	FI,SCALABLE	0	1	01
FIF_SCROLLABLE	FI,SCROLLABLE	1	2	02
FIF_REMAPPABLE	FI,REMAPPABLE	2	4	04

FreeList

16 bytes

C Include File: workbench/workbench.h
ML Include File: workbench/workbench.i

byte	hex	name	C type	ML type
0	00	fl_NumFree	WORD	WORD
2	02	fl_MemList	struct List	STRUCT

FC

see FontContents

FCH

see FontContentsHeader

FileHandle

44 bytes
(Updated for 3.0)

C Include File: dos/dosextens.h
ML Include File: dos/dosextens.i

byte	hex	C name	C type	ML Name	ML type
0	00	fh_Link	struct Message *	fh_Link	APTR
4	04	fh_Port	struct MsgPort *	fh_Interactive	APTR
8	08	fh_Type	struct MsgPort *	fh_Type	APTR
12	0C	fh_Buf	LONG	fh_Buf	LONG
16	10	fh_Pos	LONG	fh_Pos	LONG
20	14	fh_End	LONG	fh_End	LONG
24	18	fh_Func1	LONG	fh_Func1	LONG
28	1C	fh_Func2	LONG	fh_Func2	LONG
32	20	fh_Func3	LONG	fh_Func3	LONG
36	24	fh_Arg1	LONG	fh_Arg1	LONG
40	48	fh_Arg2	LONG	fh_Arg2	LONG

Note: The elements fh_Func1 and fh_Arg1 may also be referred to as fh_Funcs and fh_Args, respectively. This structure was previously found in libraries/dosextens.h (C version) and libraries/dosextens.i (ML version).

FileInfoBlock

260 bytes
(Updated for 3.0)

C Include File: dos/dos.h
ML Include File: dos/dos.i

byte	hex	name	C type	ML type
0	00	fib_DiskKey	LONG	LONG
4	04	fib_DirEntryType	LONG	LONG
8	08	fib_FileName	char[108]	STRUCT
116	74	fib_Protection	LONG	LONG
120	78	fib_EntryType	LONG	LONG
124	7C	fib_Size	LONG	LONG
128	80	fib_NumBlocks	LONG	LONG
132	84	fib_Date	struct DateStamp	STRUCT
144	90	fib_Comment	char[80]	STRUCT
224	E0	fib_OwnerUID	UWORD	UWORD
226	E2	fib_OwnerGID	UWORD	UWORD
228	E4	fib_Reserved	char[32]	STRUCT

fib_Protection

C flag name	C bit name	ML bit name	bit	decimal	hex
FIBF_DELETE	FIBB_DELETE	FIB,DELETE	0	1	01
FIBF_EXECUTE	FIBB_EXECUTE	FIB,EXECUTE	1	2	02
FIBF_WRITE	FIBB_WRITE	FIB,WRITE	2	4	04
FIBF_READ	FIBB_READ	FIB,READ	3	8	08
FIBF_ARCHIVE	FIBB_ARCHIVE	FIB,ARCHIVE	4	16	10
FIBF_PURE	FIBB_PURE	FIB,PURE	5	32	20
FIBF_SCRIPT	FIBB_SCRIPT	FIB,SCRIPT	6	64	40

Note: This structure was previously found in libraries/dos.h (C version) and libraries/dos.i (ML version).

FileLock

20 bytes

C Include File: dos/dosextens.h
ML Include File: dos/dosextens.i

byte	hex	name	C type	ML type
0	00	fl_Link	BPTR	BPTR
4	04	fl_Key	LONG	LONG
8	08	fl_Access	LONG	LONG
12	0C	fl_Task	struct MsgPort *	APTR
16	10	fl_Volume	BPTR	BPTR

Note: This structure was previously found in libraries/dosextens.h (C version) and libraries/dosextens.i (ML version).

FileRequester

56 bytes
(Updated for 3.0)

C Include File: libraries/asl.h
ML Include File: libraries/asl.i

byte	hex	name	C type	ML type
0	00	rf_Reserved0	UBYTE[4]	STRUCTURE
4	04	rf_File	STRPTR	APTR
8	08	rf_Drawer	STRPTR	APTR
12	0C	rf_Reserved1	UBYTE[10]	STRUCT
22	16	rf_LeftEdge	WORD	WORD
24	18	rf_TopEdge	WORD	WORD
26	1A	rf_Width	WORD	WORD
28	1C	rf_Height	WORD	WORD
30	1E	rf_Reserved2	UBYTE[2]	STRUCT
32	20	rf_NumArgs	LONG	LONG
36	24	rf_ArgList	struct WBArg *	APTR
40	28	rf_UserData	APTR	APTR
44	2C	rf_Reserved3	APTR	STRUCT
52	34	rf_Pattern	STRPTR	APTR

FileSysEntry

62 bytes
(New to 2.0)

C Include File: resources/filesysres.h
ML Include File: resources/filesysres.i

byte	hex	name	C type	ML type
0	00	fse_Node	struct Node	STRUCTURE
14	0E	fse_DosType	ULONG	ULONG
18	12	fse_Version	ULONG	ULONG
22	16	fse_PatchFlags	ULONG	ULONG
26	1A	fse_Type	ULONG	ULONG
30	1E	fse_Task	CPTR	CPTR
34	22	fse_Lock	BPTR	BPTR
38	26	fse_Handler	BSTR	BSTR
42	2A	fse_StackSize	ULONG	ULONG
46	2E	fse_Priority	LONG	LONG
50	32	fse_Startup	BPTR	BPTR
54	36	fse_SegList	BPTR	BPTR
58	3A	fse_GlobalVec	BPTR	BPTR

fse_PatchFlags

substitution	bit	decimal	hex
(fse_Type)	0	1	0001
(fse_Task)	1	2	0002
(fse_Lock)	2	4	0004
(fse_Handler)	3	8	0008
(fse_StackSize)	4	16	0010
(fse_Priority)	5	32	0020
(fse_Startup)	6	64	0040
(fse_SegList)	7	128	0080
(fse_GlobalVec)	8	256	0100

Note: For every entry that you wish to copy from this structure to the device node, set the appropriate bit in fse_PatchFlags. There are no names assigned to these bits in C or ML.

FileSysHeaderBlock

256 bytes
(New to 2.0)

C Include File: devices/hardblocks.h
ML Include File: devices/hardblocks.i

byte	hex	name	C type	ML type
0	00	fhb_ID	ULONG	ULONG
4	04	fhb_SummedLongs	ULONG	ULONG
8	08	fhb_ChkSum	LONG	LONG
12	0C	fhb_HostID	ULONG	ULONG

byte	hex	name	C type	ML type
16	10	fhb_Next	ULONG	ULONG
20	14	fhb_Flags	ULONG	ULONG
24	18	fhb_Reserved1	ULONG[2]	STRUCT
32	20	fhb_DosType	ULONG	ULONG
36	24	fhb_Version	ULONG	ULONG
40	28	fhb_PatchFlags	ULONG	ULONG
44	2C	fhb_Type	ULONG	ULONG
4ç8	30	fhb_Task	ULONG	ULONG
52	34	fhb_Lock	ULONG	ULONG
56	38	fhb_Handler	ULONG	ULONG
60	3C	fhb_StackSize	ULONG	ULONG
64	40	fhb_Priority	LONG	LONG
68	44	fhb_Startup	LONG	LONG
72	48	fhb_SegListBlocks	LONG	LONG
76	4C	fhb_GlobalVec	LONG	LONG
80	50	fhb_Reserved2	ULONG[23]	STRUCT
172	AC	fhb_Reserved3	ULONG[21]	STRUCT

fhb_PatchFlags

substitution	bit	decimal	hex
(fhb_Type)	0	1	0001
(fhb_Task)	1	2	0002
(fhb_Lock)	2	4	0004
(fhb_Handler)	3	8	0008
(fhb_StackSize)	4	16	0010
(fhb_Priority)	5	32	0020
(fhb_Startup)	6	64	0040
(fhb_SegList)	7	128	0080
(fhb_GlobalVec)	8	256	0100

FileSysResource

32 bytes
(New to 2.0)

C Include File: resources/filesysres.h
ML Include File: resources/filesysres.i

byte	hex	name	C type	ML type
0	00			STRUCTURE
0	00	fsr_Node	struct Node	
4	04	fsr_Creator	char *	CPTR
8	08	fsr_FileSysEntries	struct List	STRUCT

FileSysStartupMsg

16 bytes

C Include File: dos/filehandler.h
ML Include File: dos/filehandler.i

byte	hex	name	C type	ML type
0	00	fssm_Unit	ULONG	ULONG
4	04	fssm_Device	BSTR	BSTR
8	08	fssm_Environ	BPTR	BPTR
12	0C	fssm_Flags	ULONG	ULONG

FontContents

(FC in ML)
260 bytes

C Include File: diskfont/diskfont.h
ML Include File: diskfont/diskfont.i

byte	hex	name	C type	ML type
0	0000	fc_FileName	char[256]	STRUCT
256	0100	fc_YSize	UWORD	UWORD
258	0102	fc_Style	UBYTE	UBYTE
259	0103	fc_Flags	UBYTE	UBYTE

fc_Style

C flag name	C bit name	ML name	bit	decimal	hex
FSF_UNDERLINED	FSB_UNDERLINED	FS,UNDERLINED	0	1	01
FSF_BOLD	FSB_BOLD	FS,BOLD	1	2	02
FSF_ITALIC	FSB_ITALIC	FS,ITALIC	2	4	04
FSF_EXTENDED	FSB_EXTENDED	FS,EXTENDED	3	8	08

fc_Flags

C flag name	C bit name	ML name	bit	decimal	hex
FPF_ROMFONT	FPB_ROMFONT	FP,ROMFONT	0	1	01
FPF_DISKFONT	FPB_DISKFONT	FP,DISKFONT	1	2	02
FPF_REVPATH	FPB_REVPATH	FP,REVPATH	2	4	04
FPF_TALLDOT	FPB_TALLDOT	FP,TALLDOT	3	8	08
FPF_LONGDOT	FPB_LONGDOT	FP,LONGDOT	4	16	10
FPF_PROPORTIONAL	FPB_PROPORTIONAL	FP,PROPORTIONAL	5	32	20
FPF_DESIGNED	FPB_DESIGNED	FP,DESIGNED	6	64	40
FPF_REMOVED	FPB_REMOVED	FP,REMOVED	7	128	80

Note: This structure was previously found in libraries/diskfont.h (C version) and libraries/diskfont.i (ML version).

FontContentsHeader

4 bytes

C Include File: diskfont/diskfont.h
ML Include File: diskfont/diskfont.i

byte	hex	name	C type	ML type
0	00	fch_FileID	UWORD	UWORD
2	02	fch_NumEntries	UWORD	UWORD

Note: This structure was previously found in libraries/diskfont.h (C version) and libraries/diskfont.i (ML version).

FontPrefs

156 bytes
(New to 3.0)

C Include File: prefs/font.h
ML Include File: prefs/font.i

byte	hex	name	C type	ML type
0	00	fp_Reserved	LONG[3]	STRUCT
12	0C	fp_Reserved2	UWORD	UWORD
14	0E	fp_Type	UWORD	UWORD
16	10	fp_FrontPen	UBYTE	UBYTE
17	11	fp_BackPen	UBYTE	UBYTE
18	12	fp_DrawMode	UBYTE	UBYTE
20	14	fp_TextAttr	struct TextAttr	STRUCT
28	1C	fp_Name	BYTE[FONTNAMESIZE]	STRUCT

fp_Type

name	decimal	hex
FP_WBFONT	0	00
FP_SYSFONT	1	01
FP_SCREENFONT	2	02

FontRequester

24 bytes
(Updated for 3.0)

C Include File: libraries/asl.h
ML Include File: libraries/asl.i

byte	hex	C name	C type	ML name	ML type
0	00	fo_Reserved0	UBYTE[8]		
0	00			FontRequester	STRUCTURE
4	04	fo_Attr	struct TextAttr	fo_Attr	STRUCT
12	0C	fo_FrontPen	UBYTE	fo_FrontPen	UBYTE
13	0D	fo_BackPen	UBYTE	fo_BackPen	UBYTE
14	0E	fo_DrawMode	UBYTE	fo_DrawMode	UBYTE
15	0F	fo_Reserved1	UBYTE	fo_Reserved1	UBYTE
16	10	fo_UserData	APTR	fo_UserData	APTR
20	14	fo_LeftEdge	WORD	fo_LeftEdge	WORD
22	16	fo_TopEdge	WORD	fo_TopEdge	WORD
24	18	fo_Width	WORD	fo_Width	WORD
26	1A	fo_Height	WORD	fo_Height	WORD
28	1C	fo_TAttr	struct TTextAttr	fo_TAttr	STRUCT

FrameInfo

36 bytes
(New to 3.0)

C Include File: datatypes/datatypesclass.h
ML Include File: datatypes/datatypesclass.i

byte	hex	C name	C type	ML name	ML type
0	00	fri_PropertyFlags	ULONG	fri_PropertyFlags	ULONG
4	04	fri_Resolution	Point	fri_Resolution	STRUCT
8	08	fri_RedBits	UBYTE	fri_RedBits	UBYTE
9	09	fri_GreenBits	UBYTE	fri_GreenBits	UBYTE
10	0A	fri_BlueBits	UBYTE	fri_GreenBits	UBYTE
12	0C	fri_Dimensions	struct{ULONG Width; ULONG Height; ULONG Depth;}		
12	0C			fri_Width	ULONG
16	10			fri_Height	ULONG
20	14			fri_Depth	ULONG

byte	hex	C name	C type	ML name	ML type
24	18	fri_Screen	Screen *	fri_Screen	APTR
28	1C	fri_ColorMap	ColorMap *	fri_ColorMap	APTR
32	20	fri_Flags	ULONG	fri_Flags	ULONG

fri_Flags

C name	ML bit name	bit	decimal	hex
FIF_SCALABLE	FI,SCALABLE	0	1	01
FIF_SCROLLABLE	FI,SCROLLABLE	1	2	02
FIF_REMAPPABLE	FI,REMAPPABLE	2	4	04

FreeList

16 bytes

C Include File: workbench/workbench.h
ML Include File: workbench/workbench.i

byte	hex	name	C type	ML type
0	00	fl_NumFree	WORD	WORD
2	02	fl_MemList	struct List	STRUCT

Gadget

44 bytes
(Updated for 3.0)

C Include File: intuition/intuition.h
ML Include File: intuition/intuition.i

byte	hex	C name	C type	ML Name	ML type
0	00	NextGadget	struct Gadget *	gg_NextGadget	APTR
4	04	LeftEdge	WORD	gg_LeftEdge	WORD
6	06	TopEdge	WORD	gg_TopEdge	WORD
8	08	Width	WORD	gg_Width	WORD
10	0A	Height	WORD	gg_Height	WORD
12	0C	Flags	UWORD	gg_Height	WORD
14	0E	Activation	UWORD	gg_Activation	WORD
16	10	GadgetType	UWORD	gg_GadgetType	WORD
18	12	GadgetRender	APTR	gg_GadgetRender	APTR
22	16	SelectRender	APTR	gg_SelectRender	APTR
26	1A	GadgetText	struct IntuiText	gg_GadgetText	APTR

byte	hex	C name	C type	ML Name	ML type
30	1E	MutualExclude	LONG	gg_MutualExclude	LONG
34	22	SpecialInfo	APTR	gg_SpecialInfo	APTR
38	26	GadgetID	UWORD	gg_GadgetID	WORD
40	28	UserData	APTR	gg_UserData	APTR

Flags

name	bit	decimal	hex
GADGIMAGE	2	4	0004
GRELBOTTOM	3	8	0008
GRELRIGHT	4	16	0010
GRELWIDTH	5	32	0020
GRELHEIGHT	6	64	0040
SELECTED	7	128	0080
GADGDISABLED	8	256	0100

Activation

name	bit	decimal	hex
RELVERIFY	0	1	0001
GADGIMMEDIATE	1	2	0002
ENDGADGET	2	4	0004
FOLLOWMOUSE	3	8	0008
RIGHTBORDER	4	16	0010
LEFTBORDER	5	32	0020
TOPBORDER	6	64	0040
BOTTOMBORDER	7	128	0080
TOGGLESELECT	8	256	0100
STRINGCENTER	9	512	0200
STRINGRIGHT	10	1024	0400
LONGINT	11	2048	0800
ALTKEYMAP	12	4096	1000
BOOLEXTEND	13	8192	2000

GadgetInfo

58 bytes
(New to 2.0)

C Include File: intuition/cghooks.h
ML Include File: intuition/cghooks.i

byte	hex	C name	C type	ML name	ML type
0	00	gi_Screen	struct Screen *	ggi_Screen	APTR
4	04	gi_Window	struct Window *	ggi_Window	APTR
8	08	gi_Requester	struct Requester *	ggi_Requester	APTR
12	0C	gi_RastPort	struct RastPort *	ggi_RastPort	APTR
16	10	gi_Layer	struct Layer *	ggi_Layer	APTR
20	14	gi_Domain	struct IBox	ggi_Domain	STRUCT
28	1C	gi_Pens	struct {UBYTE; UBYTE;}	ggi_Pens	STRUCT
28	1C	gi_Pens.DetailPen	UBYTE		
29	1D	gi_Pens.BlockPen	UBYTE		
30	1E	gi_DrInfo	struct DrawInfo *	ggi_DrInfo	APTR
34	22	gi_Reserved	ULONG[6]		

GamePortTrigger

8 bytes

C Inclue File: devices/gameport.h
ML Include File: devices/gameport.i

byte	hex	name	type
0	00	gpt_Keys	UWORD
2	02	gpt_Timeout	UWORD
4	04	gpt_XDelta	UWORD
6	06	gpt_YDelta	UWORD

gpt_keys

C name	ML name	bit	decimal	hex
GPTF_DOWNKEYS	GPT,DOWNKEYS	0	1	01
GPTF_UPKEYS	GPT,UPKEYS	1	2	02

GelsInfo

38 bytes
(updated for 3.0)

C Include File: graphics/rastport.h
ML Include File: graphics/rastport.i

byte	hex	C name	C type	ML Name	ML type
0	00	sprRsrvd	BYTE	gi_sprRsrvd	BYTE
1	01	Flags	UBYTE	gi_sprFlags	BYTE
2	02	gelHead	struct VSprite *	gi_gelHead	APTR
6	06	gelTail	struct VSprite *	gi_gelTail	APTR

byte	hex	C name	C type	ML Name	ML type
10	0A	nextLine	WORD *	gi_nextLine	APTR
14	0E	lastColor	WORD **	gi_lastColor	APTR
18	12	collHandler	struct collTable *	gi_collTablePtr	APTR
22	16	leftmost	WORD	gi_leftmost	WORD
24	18	rightmost	WORD	gi_rightmost	WORD
26	1A	topmost	WORD	gi_topmost	WORD
28	1C	bottommost	WORD	gi_bottommost	WORD
30	1E	firstBlissObj	APTR	gi_firstBlissObj	APTR
34	22	lastBlissObj	APTR	gi_lastBlissObj	APTR

GfxBase

422 bytes
(Updated for 3.0)

C Include File: graphics/gfxbase.h
ML Include File: graphics/gfxbase.i

byte	hex	C name	C type	ML Name	ML type
0	0000	LibNode	Library	gb_LibNode	STRUCTURE
34	0022	ActiView	struct View *	gb_Actiview	APTR
38	0026	copinit	struct copinit *	gb_copinit	APTR
42	002A	cia	LONG *	gb_cia	APTR
46	002E	blitter	LONG *	gb_blitter	APTR
50	0032	LOFlist	UWORD *	gb_LOFlist	APTR
54	0036	SHFlist	UWORD *	gb_SHFlist	APTR
58	003A	blthd	struct bltnode *	gb_blthd	APTR
62	003E	bltl	struct bltnode *	gb_bltl	APTR
66	0042	bsblthd	struct bltnode *	gb_bsblthd	APTR
70	0046	bsbltl	struct bltnode *	gb_bsbltl	APTR
74	004A	vbsrv	struct Interrupt	gb_vbsrv	STRUCT
96	0060	timsrv	struct Interrupt	gb_timsrv	STRUCT
118	0076	bltsrv	struct Interrupt	gb_bltsrv	STRUCT
140	008C	TextFonts	struct List	gb_TextFonts	STRUCT
154	009A	DefaultFont	struct TextFont *	gb_DefaultFont	APTR
158	009E	Modes	UWORD	gb_Modes	UWORD
160	00A0	VBlank	BYTE	gb_VBlank	BYTE
161	00A1	Debug	BYTE	gb_Debug	BYTE
162	00A2	BeamSync	WORD	gb_BeamSync	UWORD
164	00A4	systembplcon0	WORD	gb_systembplcon0	WORD
166	00A6	SpriteReserved	UBYTE	gb_SpriteReserved	BYTE
167	00A7	bytereserved	UBYTE	gb_bytereserved	BYTE
168	00A8	Flags	UWORD	gb_Flags	WORD
170	00AA	BlitLock	WORD	gb_BlitLock	WORD

byte	hex	C name	C type	ML Name	ML type
172	00AC	BlitNest	WORD	gb_BlitNest	WORD
174	00AE	BlitWaitQ	struct List	gb_BlitWaitQ	STRUCT
188	00BC	BlitOwner	struct Task *	gb_BlitOwner	APTR
192	00C0	TOFWaitQ	struct List	gb_TOFWaitQ	STRUCT
206	00CE	DisplayFlags	UWORD	gb_DisplayFlags	WORD
208	00D0	SimpleSprites	struct SimpleSprite *	gb_SimpleSprites	APTR
212	00D4	MaxDisplayRow	UWORD	gb_MaxDisplayRow	WORD
214	00D6	MaxDisplayColumn	UWORD	gb_MaxDisplayColumn	WORD
216	00D8	NormalDisplayRows	UWORD	gb_NormalDisplayRows	WORD
218	00DA	NormalDisplayColumns	UWORD	gb_NormalDisplayColumns	WORD
220	00DC	NormalDPMX	UWORD	gb_NormalDPMX	WORD
222	00DE	NormalDPMY	UWORD	gb_NormalDPMY	WORD
224	00E0	LastChanceMemory	struct SignalSemaphore *	gb_LastChanceMemory	APTR
228	00E4	LCMptr	UWORD *	gb_LCMptr	APTR
232	00E8	MicrosPerLine	UWORD	gb_MicrosPerLine	WORD
234	00EA	MinDisplayColumn	UWORD	gb_MinDisplayColumn	WORD
236	00EC	ChipRevBits0	UBYTE	gb_ChipRevBits0	UBYTE
237	00ED	crb_reserved	UBYTE[5]	gb_crb_reserved	STRUCT
242	00F2	monitor_id	UWORD	gb_monitor_id	STRUCT
244	00F4	hedley	ULONG[8]	gb_hedley	STRUCT
276	0114	hedley_sprites	ULONG[8]	gb_hedley_sprites	STRUCT
308	0134	hedley_sprites1	ULONG[8]	gb_hedley_sprites1	STRUCT
340	0154	hedley_count	WORD	gb_hedley_count	WORD
342	0156	hedley_flags	UWORD	gb_hedley_flags	WORD
344	0158	hedley_tmp	WORD	gb_hedley_tmp	WORD
346	015A	hash_table	LONG *	gb_hash_table	APTR
350	015E	current_tot_rows	UWORD	gb_current_tot_rows	UWORD
352	0160	current_tot_cclks	UWORD	gb_current_tot_cclks	UWORD
354	0162	hedley_hint	UBYTE	gb_hedley_hint	UBYTE
355	0163	hedley_hint2	UBYTE	gb_hedley_hint2	UBYTE
356	0164	nreserved	ULONG[4]	gb_nreserved	STRUCT
372	0174	a2024_sync_raster	LONG *	gb_a2024_sync_raster	APTR
376	0178	control_delta_pal	WORD	b_control_delta_pal	WORD
378	017A	control_delta_ntsc	WORD	gb_control_delta_ntsc	WORD
380	017C	current_monitor	struct MonitorSpec *	gb_current_monitor	APTR
384	0180	MonitorList	struct List	gb_MonitorList	STRUCT
398	018E	default_monitor	struct MonitorSpec *	gb_default_monitor	APTR
402	0192	MonitorListSemaphore	struct SignalSemaphore *	gb_MonitorListSemaphore	APTR
406	0196	DisplayInfoDataBase	VOID *	gb_DisplayInfoDataBase	APTR
410	019A	lapad	WORD	lapad	WORD
412	019C	ActiViewCprSemaphore	struct SignalSemaphore *	gb_ActiViewCprSemaphore	APTR
416	01A0	UtilityBase	ULONG *	gb_UtilityBase	APTR
420	01A4	ExecBase	ULONG *	gb_ExecBase	APTR

GlyphEngine

8 bytes
(New to 3.0)

C Include File: diskfont/glyph.h
ML Include File: diskfont/glyph.i

byte	hex	name	C type	ML type
0	00	gle_Library	struct Library *	APTR
4	04	gle_Name	char *	APTR

GlyphMap

36 bytes
(New to 3.0)

C Include File: diskfont/glyph.h
ML Include File: diskfont/glyph.i

byte	hex	name	C type	ML type
0	00	glm_BMModulo	UWORD	UWORD
2	02	glm_BMRows	UWORD	UWORD
4	04	glm_BlackLeft	UWORD	UWORD
6	06	glm_BlackTop	UWORD	UWORD
8	08	glm_BlackWidth	UWORD	UWORD
10	0A	glm_BlackHeight	UWORD	UWORD
12	0C	glm_XOrigin	FIXED	ULONG
16	10	glm_YOrigin	FIXED	ULONG
20	14	glm_X0	WORD	WORD
22	16	glm_Y0	WORD	WORD
24	18	glm_X1	WORD	WORD
26	1A	glm_Y1	WORD	WORD
28	1C	glm_Width	FIXED	ULONG
32	20	glm_BitMap	UBYTE *	APTR

GlyphWidthEntry

14 bytes
(New to 3.0)

byte	hex	name	C type	ML type
0	00	gwe_Node	struct MinNode	STRUCT
8	08	gwe_Code	UWORD	UWORD
10	0A	gwe_Width	FIXED	ULONG

gpGoInactive

8 bytes
(New to 2.0)

C Include File: intuition/gadgetclass.h
ML Include File: intuition/gadgetclass.i

byte	hex	C name	C type	ML name	ML type
0	00	MethodID	ULONG		
4	04	gpgi_GInfo	struct GadgetInfo *	gpgi_GInfo	APTR

gpHitTest

12 bytes
(New to 2.0)

C Include File: intuition/gadgetclass.h
ML Include File: intuition/gadgetclass.i

byte	hex	C name	C type	ML name	ML type
0	00	MethodID	ULONG		
4	04	gpht_GInfo	struct GadgetInfo *	gpht_GInfo	APTR
8	08	gpht_Mouse	struct {WORD; WORD;}		
8	08	gpht_Mouse.X	WORD	gpht_MouseX	WORD
10	0A	gpht_Mouse.Y	WORD	gpht_MouseY	WORD

gpInput

20 bytes
(New to 2.0)

C Include File: intuition/gadgetclass.h
ML Include File: intuition/gadgetclass.i

byte	hex	C name	C type	ML name	ML type
0	00	MethodID	ULONG		
4	04	gpi_GInfo	struct GadgetInfo	gpi_GInfo	APTR
8	08	gpi_IEvent	struct InputEvent *	gpi_IEvent	APTR
12	0C	gpi_Termination	LONG *	gpi_Termination	APTR
16	10	gpht_Mouse	struct {WORD; WORD;}		
16	10	gpht_Mouse.X	WORD	gpht_MouseX	WORD
18	12	gpht_Mouse.Y	WORD	gpht_MouseY	WORD

gpLayout

12 bytes
(New to 3.0)

C Include File: intuition/gadgetclass.h
ML Include File: intuition/gadgetclass.i

byte	hex	C name	C type	ML name	ML type
0	00	MethodID	ULONG		
4	04	gpl_GInfo	struct GadgetInfo *	gpl_GInfo	APTR
8	08	gpl_Initial	ULONG	gpl_Initial	APTR

gpRender

16 bytes
(New to 2.0)

C Include File: intuition/gadgetclass.h
ML Include File: intuition/gadgetclass.i

byte	hex	C name	C type	ML name	ML type
0	00	MethodID	ULONG		
4	04	gpr_GInfo	struct GadgetInfo *	gpr_GInfo	APTR
8	08	gpr_RPort	struct RastPort *	gpr_RPort	APTR
12	0C	gpr_Redraw	LONG	gpr_Redraw	LONG

gpr_Redraw

flag name	decimal	hex
GREDRAW_UPDATE	2	02
GREDRAW_REDRAW	1	01
GREDRAW_TOGGLE	0	00

Hook

20 bytes
(New to 2.0)

C Include File: utility/hook.h
ML Include File: utility/hook.i

byte	hex	C name	C type	ML name	ML type
0	00	h_MinNode	struct MinNode		
8	08	h_Entry	ULONG(*)()	h_Entry	APTR
12	0C	h_SubEntry	ULONG(*)()	h_SubEntry	APTR
16	10	h_Data	VOID *	h_Data	APTR

IBox

8 bytes
(New to 2.0)

C Include File: intuition/intuition.h
ML Include File: intuition/intuition.i

byte	hex	C name	C type	ML name	ML type
0	00	Left	WORD	ibox_Left	WORD
2	02	Top	WORD	ibox_Top	WORD
4	04	Width	WORD	ibox_Width	WORD
6	06	Height	WORD	ibox_Height	WORD

IClass

(ICLASS in ML)
52 bytes
(New to 2.0)

C Include File: intuition/classes.h
ML Include File: intuition/classes.i

byte	hex	name	C type	ML type
0	00	cl_Dispatcher	struct Hook	STRUCT
20	14	cl_Reserved	ULONG	ULONG
24	18	cl_Super	struct IClass *	APTR
28	1C	cl_ID	ClassID	APTR
32	20	cl_InstOffset	UWORD	UWORD
34	22	cl_InstSize	UWORD	UWORD
36	24	cl_UserData	ULONG	ULONG
40	28	cl_SubclassCount	ULONG	ULONG
44	2C	cl_ObjectCount	ULONG	ULONG
48	30	cl_Flags	ULONG	ULONG

cl_Flags

flag name	ML bit name	bit	decimal	hex
CLF_INLIST	CLB_INLIST	0	1	01

IControlPrefs

28 bytes
(New to 3.0)

C Include File: prefs/icontrol.i
ML Include File: prefs/icontrol.h

byte	hex	name	C type	ML type
0	00	ic_Reserved	LONG[4]	STRUCT
16	10	ic_TimeOut	UWORD	UWORD
18	12	ic_MetaDrag	WORD	WORD
20	14	ic_Flags	ULONG	ULONG
24	18	ic_WBtoFront	UBYTE	UBYTE
25	19	ic_FrontToBack	UBYTE	UBYTE
26	1A	ic_ReqTrue	UBYTE	UBYTE
27	1B	ic_ReqFalse	UBYTE	UBYTE

ic_Flags

C flag name	C bit name	ML bit name	bit	decimal	hex
ICF_COERCE_COLORS	ICB_COERCE_COLORS	IC,COERCE_COLORS	0	1	01
ICF_COERCE_LACE	ICB_COERCE_LACE	IC,COERCE_LACE	1	2	02
ICF_STRGAD_FILTER	ICB_STRGAD_FILTER	IC,STRGAD_FILTER	2	4	04
ICF_MENUSNAP	ICB_MENUSNAP	IC,MENUSNAP	3	8	08

IENewTablet

32 bytes
(New to 3.0)

C Include File: devices/inputevent.h
ML Include File: devices/inputevent.i

byte	hex	name	C type	ML type
0	00	ient_CallBack	struct Hook *	APTR
4	04	ient_ScaledX	UWORD	UWORD
6	06	ient_ScaledY	UWORD	UWORD
8	08	ient_ScaledXFraction	UWORD	UWORD
10	0A	ient_ScaledYFraction	UWORD	UWORD
12	0C	ient_TabletX	ULONG	ULONG

byte	hex	name	C type	ML type
16	10	ient_TabletY	ULONG	ULONG
20	14	ient_RangeX	ULONG	ULONG
24	18	ient_RangeY	ULONG	ULONG
28	1C	ient_TagList	struct TagItem *	APTR

IEPointerPixel

8 bytes
(New to 2.0)

C Include File: devices/inputevent.h
ML Include File: devices/inputevent.i

byte	hex	C name	C type	ML name	ML type
0	00	iepp_Screen	struct Screen *	iepp_Screen	APTR
4	04	iepp_Position	struct {WORD; WORD;}	iepp_Position	LABEL
4	04	iepp_Position.X	WORD	iepp_PositionX	WORD
6	06	iepp_Position.Y	WORD	iepp_PositionY	WORD

IEPointerTablet

10 bytes
(New to 2.0)

C Include File: devices/inputevent.h
ML Include File: devices/inputevent.i

byte	hex	C name	C type	ML name	ML type
0	00	iept_Range	struct {UWORD; UWORD;}	iept_Range	LABEL
0	00	iept_Range.X	UWORD	iept_RangeX	UWORD
2	02	iept_Range.Y	UWORD	iept_RangeY	UWORD
4	04	iept_Value	struct {UWORD; UWORD;}	iept_Value	LABEL
4	04	iept_Value.X	UWORD	iept_ValueX	UWORD
6	06	iept_Value.Y	UWORD	iept_ValueY	UWORD
8	08	iept_Pressure	WORD	iept_Pressure	WORD

IFFHandle

12 bytes
(New to 2.0)

C Include File: libraries/iffparse.h
ML Include File: libraries/iffparse.i

byte	hex	name	type
0	00	iff_Stream	ULONG
4	04	iff_Flags	ULONG
8	08	iff_Depth	LONG

iff_Flags

name	decimal	hex
IFFF_READ	0	00000000
IFFF_WRITE	1	00000001
IFFF_RWBITS	1	00000001
IFFF_FSEEK	2	00000002
IFFF_RSEEK	4	00000004
IFFF_RESERVED	4294901760	FFFF0000

IFFStreamCmd

12 bytes
(New to 2.0)

C Include File: libraries/iffparse.h
ML Include File: libraries/iffparse.i

byte	hex	C name	C type	ML name	ML type
0	00	sc_Command	LONG	isc_Command	LONG
4	04	sc_Buf	APTR	isc_Buf	APTR
8	08	sc_NBytes	LONG	isc_NBytes	LONG

Image

20 bytes
(Updated for 3.0)

C Include File: intuition/intuition.h
ML Include File: intuition/intuition.i

byte	hex	C name	C type	ML name	ML type
0	00	LeftEdge	WORD	ig_LeftEdge	WORD
2	02	TopEdge	WORD	ig_TopEdge	WORD
4	04	Width	WORD	ig_Width	WORD
6	06	Height	WORD	ig_Height	WORD
8	08	Depth	WORD	ig_Depth	WORD
10	0A	ImageData	UWORD *	ig_ImageData	APTR
14	0E	PlanePick	UBYTE	ig_PlanePick	BYTE
15	0F	PlaneOnOff	UBYTE	ig_PlaneOnOff	BYTE
16	10	NextImage	struct Image *	ig_NextImage	APTR

ImpDraw

24 bytes
(New to 2.0)

C Include File: intuition/imageclass.h
ML Include File: intuition/imageclass.i

byte	hex	C name	C type	ML name	ML type
0	00	MethodID	ULONG		
4	04	imp_RPort	struct RastPort *	impd_RPort	APTR
8	08	imp_Offset	struct {WORD; WORD;}		
8	08	imp_Offset.X	WORD	impd_OffsetX	WORD
10	0A	imp_Offset.Y	WORD	impd_OffsetY	WORD
12	0C	imp_State	ULONG	impd_State	ULONG
16	10	imp_DrInfo	struct DrawInfo *	impd_DrInfo	APTR
20	14	imp_Dimensions	struct {WORD; WORD;}		
20	14	imp_Dimensions.Width	WORD	impd_DimensionsWidth	WORD
22	16	imp_Dimensions.Height	WORD	impd_DimensionsHeight	WORD

ImpErase

16 bytes
(New to 2.0)

C Include File: intuition/imageclass.h
ML Include File: intuition/imageclass.i

byte	hex	C name	C type	ML name	ML type
0	00	MethodID	ULONG		
4	04	imp_RPort	struct RastPort *	impe_RPort	APTR
8	08	imp_Offset	struct {WORD; WORD;}		
8	08	imp_Offset.X	WORD	impe_OffsetX	WORD
10	0A	imp_Offset.Y	WORD	impe_OffsetY	WORD
12	0C	imp_Dimensions	struct {WORD; WORD;}		
12	0C	imp_Dimensions.Width	WORD	impe_DimensionsWidth	WORD
14	0E	imp_Dimensions.Height	WORD	impe_DimensionsHeight	WORD

ImpFrameBox

20 bytes
(New to 2.0)

C Include File: intuition/imageclass.h
ML Include File: intuition/inageclass.i

byte	hex	C name	C type	ML name	ML type
0	00	MethodID	ULONG		
4	04	imp_ContentsBox	struct IBox *	impf_ContentsBox	APTR
8	08	imp_FrameBox	struct IBox *	impf_FrameBox	APTR
12	0C	imp_DrInfo	struct DrawInfo *	impf_DrInfo	APTR
16	10	imp_FrameFlags	ULONG	impf_FrameFlags	LONG

ImpHitTest

12 bytes
(New to 2.0)

C Include File: intuition/imageclass.h
ML Include File: intuition/imageclass.i

byte	hex	C name	C type	ML name	ML type
0	00	MethodID	ULONG		
4	04	imp_Point	struct {WORD; WORD;}		
4	04	imp_Point.X	WORD	imph_PointX	WORD
6	06	imp_Point.Y	WORD	imph_PointY	WORD
8	08	imp_Dimensions	struct {WORD; WORD;}		
8	08	imp_Dimensions.Width	WORD	imph_DimensionsWidth	WORD
10	0A	imp_Dimensions.Height	WORD	imph_DimensionsHeight	WORD

InfoData

36 bytes
(Updated for 3.0)

C Include File: dos/dos.h
ML Include File: dos/dos.i

byte	hex	name	type
0	00	id_NumSoftErrors	LONG
4	04	id_UnitNumber	LONG
8	08	id_DiskState	LONG
12	0C	id_NumBlocks	LONG
16	10	id_NumBlocksUsed	LONG
20	14	id_BytesPerBlock	LONG
24	18	id_DiskType	LONG
28	1C	id_VolumeNode	BPTR
32	20	id_InUse	LONG

id_DiskState

name	decimal	hex
ID_WRITE_PROTECTED	128	80
ID_VALIDATING	129	81
ID_VALIDATED	130	82

id_DiskType

name	decimal	hex
ID_NO_DISK_PRESENT	-1	FFFFFFFF
ID_UNREADABLE_DISK	1111573504	42414400
ID_DOS_DISK	1146049280	444F5300
ID_FFS_DISK	1146049281	444F5301
ID_INTER_DOS_DISK	1146049282	444F5302
ID_INTER_FFS_DISK	1146049283	444F5303
ID_FASTDIR_DOS_DISK	1146049284	444F5304
ID_FASTDIR_FFS_DISK	1146049285	444F5305
ID_KICKSTART_DISK	1263092555	4B49434B
ID_MSDOS_DISK	1297302528	4D534400
ID_NOT_REALLY_DOS	1313099603	4E444F53

Note: This structure was previously found in libraries/dos.h (C version) and libraries/dos.i (ML version).

InputEvent

22 bytes

C Include File: devices/inputevent.h
ML Include File: devices/inputevent.i

byte	hex	name	C type	ML type
0	00	ie_NextEvent	struct InputEvent *	APTR
4	04	ie_Class	UBYTE	UBYTE
5	05	ie_SubClass	UBYTE	UBYTE
6	06	ie_Code	UWORD	UWORD
8	08	ie_Qualifier	UWORD	UWORD
10	0A	ie_X	WORD	WORD
12	0C	ie_Y	WORD	WORD
14	0E	ie_TimeStamp	struct timeval	STRUCT

ie_Code

name	decimal	hex
IECODE_KEY_CODE_FIRST	0	00
IECODE_C0_FIRST	0	00
IECODE_NEWACTIVE	1	01
IECODE_NEWSIZE	2	02
IECODE_REFRESH	3	03
IECODE_C0_LAST	31	1F
IECODE_ASCII_FIRST	32	20
IECODE_LBUTTON	104	68
IECODE_RBUTTON	105	69
IECODE_MBUTTON	106	6A
IECODE_KEY_CODE_LAST	119	77
IECODE_COMM_CODE_FIRST	120	78
IECODE_ASCII_LAST	126	7E
IECODE_COMM_CODE_LAST	127	7F
IECODE_ASCII_DEL	127	7F
IECODE_UP_PREFIX	128	80
IECODE_C1_FIRST	128	80
IECODE_C1_LAST	159	9F
IECODE_LATIN1_FIRST	160	A0
IECODE_LATIN2_LAST	255	FF
IECODE_NOBUTTON	255	FF

ie_Qualifier

flag name	bit name	bit	decimal	hex
IEQUALIFIER_LSHIFT	IEQUALIFIERB_LSHIFT	0	1	0001
IEQUALIFIER_RSHIFT	IEQUALIFIERB_RSHIFT	1	2	0002
IEQUALIFIER_CAPSLOCK	IEQUALIFIERB_CAPSLOCK	2	4	0004
IEQUALIFIER_CONTROL	IEQUALIFIERB_CONTROL	3	8	0008
IEQUALIFIER_LALT	IEQUALIFIERB_LALT	4	16	0010
IEQUALIFIER_RALT	IEQUALIFIERB_RALT	5	32	0020
IEQUALIFIER_LCOMMAND	IEQUALIFIERB_LCOMMAND	6	64	0040
IEQUALIFIER_RCOMMAND	IEQUALIFIERB_RCOMMAND	7	128	0080
IEQUALIFIER_NUMERICPAD	IEQUALIFIERB_NUMERICPAD	8	256	0100
IEQUALIFIER_REPEAT	IEQUALIFIERB_REPEAT	9	512	0200
IEQUALIFIER_INTERRUPT	IEQUALIFIERB_INTERRUPT	10	1024	0400
IEQUALIFIER_MULTIBROADCAST	IEQUALIFIERB_MULTIBROADCAST	11	2048	0800
IEQUALIFIER_MIDBUTTON	IEQUALIFIERB_MIDBUTTON	12	4096	1000
IEQUALIFIER_RBUTTON	IEQUALIFIERB_RBUTTON	13	8192	2000
IEQUALIFIER_LEFTBUTTON	IEQUALIFIERB_LEFTBUTTON	14	16384	4000
IEQUALIFIER_RELATIVEMOUSE	IEQUALIFIERB_RELATIVEMOUSE	15	32768	8000

InputPrefs

44 bytes
(New to 3.0)

C Include File: prefs/input.h
ML Include File: prefs/input.i

byte	hex	name	C type	ML type
0	00	ip_Keymap	char[16]	STRUCT
16	10	ip_PointerTicks	UWORD	UWORD
18	12	ip_DoubleClick	struct timeval	STRUCT
26	1A	ip_KeyRptDelay	struct timeval	STRUCT
34	22	ip_KeyRptSpeed	struct timeval	STRUCT
42	2A	ip_MouseAccel	WORD	WORD

InputXpression

12 bytes
(New to 2.0)

C Include File: libraries/commodities.h
ML Include File: libraries/commodities.i

byte	hex	name	type
0	00	ix_Version	UBYTE
1	01	ix_Class	UBYTE
2	02	ix_Code	UWORD
4	04	ix_CodeMask	UWORD
6	06	ix_Qualifier	UWORD
8	08	ix_QualMask	UWORD
10	0A	ix_QualSame	UWORD

ix_Version

name	decimal	hex
IX_VERSION	2	02

ix_QualSame

name	decimal	hex
IXSYM_SHIFT	1	01
IXSYM_CAPS	2	02
IXSYM_ALT	4	04

ix_QualMask

name	decimal	hex
IX_NORMALQUALS	32767	7FFF

Interrupt

(IS in ML)
22 bytes

C Include File: exec/interrupts.h
ML Include File: exec/interrupts.i

byte	hex	name	C type	ML name	ML type
0	00	is_Node	struct Node		STRUCTURE
14	0E	is_Data	APTR	IS_DATA	APTR
18	12	is_Code	VOID(*)()	IS_CODE	APTR

IntuiMessage

52 bytes

C Include File: intuition/intuition.h
ML Include File: intuition/intuition.i

hex	byte	C name	C type	ML name	ML type
0	00	ExecMessage	struct Message	im_ExecMessage	STRUCT
20	14	Class	ULONG	im_Class	LONG
24	18	Code	UWORD	im_Code	WORD
26	1A	Qualifier	UWORD	im_Qualifier	WORD
28	1C	IAddress	APTR	im_IAddress	APTR
32	20	MouseX	WORD	im_MouseX	WORD
34	22	MouseY	WORD	im_MouseY	WORD
36	24	Seconds	ULONG	im_Seconds	LONG
40	28	Micros	ULONG	im_Micros	LONG
44	2C	IDCMPWindow	struct Window *	im_IDCMPWindow	APTR
48	30	SpecialLink	struct IntuiMessage *	im_SpecialLink	APTR

IntuiText

20 bytes
(Updated for 3.0)

C Include File: intuition/intuition.h
ML Include File: intuition/intuition.i

byte	hex	C name	C type	ML name	ML type
0	00	FrontPen	UBYTE	it_FrontPen	BYTE
1	01	BackPen	UBYTE	it_BackPen	BYTE
2	02	DrawMode	UBYTE	it_DrawMode	BYTE
3	03			it_KludgeFill00	BYTE
4	04	LeftEdge	WORD	it_LeftEdge	WORD
6	06	TopEdge	WORD	it_TopEdge	WORD
8	08	ITextFont	struct TextAttr *	it_ITextFont	APTR
12	0C	IText	UBYTE *	it_IText	APTR
16	10	NextText	struct IntuiText *	it_NextText	APTR

IntuitionBase

80 bytes

C Include File: intuition/intuitionbase.h
ML Include File: intuition/intuitionbase.i

byte	hex	C name	C type	ML name	ML type
0	00	LibNode	Library	ib_LibNode	STRUCT
34	22	ViewLord	View	ib ViewLord	STRUCT
52	34	ActiveWindow	struct Window *	ib_ActiveWindow	APTR
56	38	ActiveScreen	struct Screen *	ib_ActiveScreen	APTR
60	3C	FirstScreen	struct Screen *	ib_FirstScreen	APTR
64	40	Flags	ULONG	ib_Flags	ULONG
68	44	MouseY	WORD	ib_MouseY	WORD
70	46	MouseX	WORD	ib_MouseX	WORD
72	48	Seconds	ULONG	ib_Seconds	ULONG
76	4C	Micros	ULONG	ib_Micros	ULONG

IntVector

(IV in ML)
12 bytes

C Include File: exec/interrupts.h
ML Include File: exec/interrupts.i

byte	hex	C name	C type	ML name	ML type
0	00	iv_Data	APTR	IV_DATA	APTR
4	04	iv_Code	VOID(*)()	IV_CODE	APTR
8	08	iv_Node	struct Node *	IV_NODE	APTR

IO

see IORequest

IOAudio

68 bytes

C Include File: devices/audio.h
ML Include File: devices/audio.i

byte	hex	name	C type	ML type
0	00	ioa_Request	struct IORequest	STRUCTURE
32	20	ioa_AllocKey	WORD	WORD
34	22	ioa_Data	UBYTE *	APTR
38	26	ioa_Length	ULONG	ULONG
42	2A	ioa_Period	UWORD	UWORD
44	2C	ioa_Volume	UWORD	UWORD
46	2E	ioa_Cycles	UWORD	UWORD
48	30	ioa_WriteMsg	struct Message	STRUCT

IoBuff

256 bytes
(New to 2.0)

C Include File: rexx/rexxio.h
ML Include File: rexx/rexxio.i

byte	hex	C name	C type	ML name	ML type
0	00				STRUCTURE
0	00	iobNode	struct RexxRsrc		
32	20	iobRpt	APTR	iobRpt	APTR
36	24	iobRct	LONG	iobRct	LONG
40	28	iobDFH	LONG	iobDFH	LONG
44	2C	iobLock	APTR	iobLock	APTR
48	30	iobBct	LONG	iobBct	LONG
52	34	iobArea	BYTE[RXBUFFSZ]	iobArea	STRUCT

IOClipReq

52 bytes

C Include File: devices/clipboard.h
ML Include File: devices/clipboard.i

byte	hex	name	C type	ML type
0	00	io_Message	struct Message	STRUCT
20	14	io_Device	struct Device *	APTR
24	18	io_Unit	struct Unit *	APTR
28	1C	io_Command	UWORD	UWORD
30	1E	io_Flags	UBYTE	UBYTE
31	1F	io_Error	BYTE	BYTE
32	20	io_Actual	ULONG	ULONG
36	24	io_Length	ULONG	ULONG
40	28	io_Data	STRPTR	APTR
44	2C	io_Offset	ULONG	ULONG
48	30	io_ClipID	LONG	LONG

IODRPReq

62 bytes
(Updated for 3.0)

C Include File: devices/printer.h
ML Include File: devices/printer.i

byte	hex	name	C type	ML type
0	00	io_Message	struct Message	STRUCTURE
20	14	io_Device	struct Device *	
24	18	io_Unit	struct Unit *	
28	1C	io_Command	UWORD	
30	1E	io_Flags	UBYTE	
31	1F	io_Error	BYTE	
32	20	io_RastPort	struct RastPort *	APTR
36	24	io_ColorMap	struct ColorMap *	APTR
40	28	io_Modes	ULONG	ULONG
44	2C	io_SrcX	UWORD	UWORD
46	2E	io_SrcY	UWORD	UWORD
48	30	io_SrcWidth	UWORD	UWORD
50	32	io_SrcHeight	UWORD	UWORD
52	34	io_DestCols	LONG	LONG
56	38	io_DestRows	LONG	LONG
60	3C	io_Special	UWORD	UWORD

io_Special

name	decimal	hex
SPECIAL_MILCOLS	1	0001
SPECIAL_MILROWS	2	0002
SPECIAL_FULLCOLS	4	0004
SPECIAL_FULLROWS	8	0008

name	decimal	hex
SPECIAL_FRACCOLS	16	0010
SPECIAL_FRACROWS	32	0020
SPECIAL_CENTER	64	0040
SPECIAL_ASPECT	128	0080
SPECIAL_DENSITY1	256	0100
SPECIAL_DENSITY2	512	0200
SPECIAL_DENSITY3	768	0300
SPECIAL_DENSITY4	1024	0400
SPECIAL_DENSITY5	1280	0500
SPECIAL_DENSITY6	1536	0600
SPECIAL_DENSITY7	1792	0700
SPECIAL_NOFORMFEED	2048	0800
SPECIAL_TRUSTME	4096	1000
SPECIAL_NOPRINT	8192	2000

io_Error

name	decimal	hex
PDERR_NOERR	0	00
PDERR_CANCEL	1	01
PD_NOTGRAPHICS	2	02
PD_INVERTHAM	3	03
PD_BADDIMENSION	4	04
PD_DIMENSIONOVFLOW	5	05
PD_INTERNALMEMORY	6	06
PD_BUFFERMEMORY	7	07

IOExtPar

62 bytes
(Updated for 3.0)

C Include File: devices/parallel.h
ML Include File: devices/parallel.i

byte	hex	name	C type	ML name	ML type
0	00	IOPar	struct IOStdReq		STRUCTURE
48	30	io_PExtFlags	ULONG	IO_PEXTFLAGS	ULONG
52	34	io_Status	UBYTE	IO_PARSTATUS	UBYTE
53	35	io_ParFlags	UBYTE	IO_PARFLAGS	UBYTE
54	36	io_PTermArray	struct IOPArray	IO_PTERMARRAY	STRUCT

io_Status

C flag name	C bit name	ML bit name	bit	decimal	hex
IOPTF_PARBUSY	IOPTB_PARBUSY	IOPT,PARBUSY	0	1	01
IOPTF_PAPEROUT	IOPTB_PAPEROUT	IOPT,PAPEROUT	1	2	02
IOPTF_PARSEL	IOPTB_PARSEL	IOPT,PARSEL	2	4	04
C flag name	**C bit name**	**ML bit name**	**bit**	**decimal**	**hex**
IOPTF_RWDIR	IOPTB_RWDIR	IOPT,RWDIR	3	8	08
IOPARF_ACTIVE	IOPARB_ACTIVE	IOPAR,ACTIVE	4	16	10
IOPARF_ABORT	IOPARB_ABORT	IOPAR,ABORT	5	32	20
IOPARF_QUEUED	IOPARB_QUEUED	IOPAR,QUEUED	6	64	40

io_ParFlags

C flag name	C bit name	ML bit name	bit	decimal	hex
PARF_EOFMODE	PARB_EOFMODE	PAR,EOFMODE	1	2	02
PARF_ACKMODE	PARB_ACKMODE	PAR,ACKMODE	2	4	04
PARF_RADBOOGIE	PARB_RADBOOGIE	PAR,RADBOOGIE	3	8	08
PARF_FASTMODE	PARB_FASTMODE	PAR,FASTMODE	3	8	08
PARF_SLOWMODE	PARB_SLOWMODE	PAR,SLOWMODE	4	16	10
PARF_SHARED	PARB_SHARED	PAR,SHARED	5	32	20

IOExtSer

(IOEXTSER in ML)
82 bytes

C Include File: devices/serial.h
ML Include File: devices/serial.i

byte	hex	name	C type	ML name	ML type
0	00	IOSer	struct IOStdReq		STRUCTURE
48	30	io_CtlChar	ULONG	IO_CTLCHAR	ULONG
52	34	io_RBufLen	ULONG	IO_RBUFLEN	ULONG
56	38	io_ExtFlags	ULONG	IO_EXTFLAGS	ULONG
60	3C	io_Baud	ULONG	IO_BAUD	ULONG
64	40	io_BrkTime	ULONG	IO_BRKTIME	ULONG
68	44	io_TermArray	struct IOTArray	IO_TERMARRAY	STRUCT
76	4C	io_ReadLen	UBYTE	IO_READLEN	UBYTE
77	4D	io_WriteLen	UBYTE	IO_WRITELEN	UBYTE
78	4E	io_StopBits	UBYTE	IO_STOPBITS	UBYTE
79	4F	io_SerFlags	UBYTE	IO_SERFLAGS	UBYTE
80	50	io_Status	UWORD	IO_STATUS	UWORD

io_ExtFlags

C flag name	C bit name	ML bit name	bit	decimal	hex
SEXTF_MARK	SEXTB_MARK	SEXT,MARK	0	1	01
SEXTF_MSPON	SEXTB_MSPON	SEXT,MSPON	1	2	02

io_SerFlags

C flag name	C bit name	ML bit name	bit	decimal	hex
SERF_PARTY_ON	SERB_PARTY_ON	SER,PARTY_ON	0	1	01
SERF_PARTY_ODD	SERB_PARTY_ODD	SER,PARTY_ODD	1	2	02
SERF_7WIRE	SERB_7WIRE	SER,7WIRE	2	4	04
SERF_QUEUEDBRK	SERB_QUEUEDBRK	SER,QUEUEDBRK	3	8	08
SERF_RAD_BOOGIE	SERB_RAD_BOOGIE	SER,RAD_BOOGIE	4	16	10
SERF_SHARED	SERB_SHARED	SER,SHARED	5	32	20
SERF_EOFMODE	SERB_EOFMODE	SER,EOFMODE	6	64	40
SERF_XDISABLED	SERB	SER,XDISABLED	7	128	80

io_Status

C flag name	C bit name	ML name	bit	decimal	hex
IOSTF_OVERRUN	IOSTB_OVERRUN	IOST,OVERRUN	8	256	0100
IOSTF_WROTEBROKE	IOSTB_WROTEBROKE	IOST,WROTEBROKE	9	512	0200
IOSTF_READBREAK	IOSTB_READBREAK	IOST,READBREAK	10	1024	0400
IOSTF_XOFFWRITE	IOSTB_XOFFWRITE	IOST,XOFFWRITE	11	2048	0800
IOSTF_XOFFREAD	IOSTB_XOFFREAD	IOST,XOFFREAD	12	4096	1000

IOExtTD

56 bytes

C Include File: devices/trackdisk.h
ML Include File: devices/trackdisk.i

byte	hex	name	C type	ML name	ML type
0	00	iotd_Req	struct IOStdReq		STRUCTURE
48	30	iotd_Count	ULONG	IOTD_COUNT	ULONG
52	34	iotd_SecLabel	ULONG	IOTD_SECLABEL	ULONG

IOPArray

(PTERMARRAY in ML)
8 bytes

C Include File: devices/parallel.h
ML Include File: devices/parallel.i

byte	hex	C name	C type	ML name	ML type
0	00	PTermArray0	ULONG	PTERMARRAY_0	ULONG
4	04	PTermArray1	ULONG	PTERMARRAY_1	ULONG

IOPrtCmdReq

38 bytes

C Include File: devices/printer.h
ML Include File: devices/printer.i

byte	hex	name	C type	ML type
0	00	io_Message	struct Message	STRUCTURE
20	14	io_Device	struct Device *	
24	18	io_Unit	struct Unit *	
28	1C	io_Command	UWORD	
30	1E	io_Flags	UBYTE	
31	1F	io_Error	BYTE	
32	20	io_PrtCommand	UWORD	UWORD
34	22	io_Parm0	UBYTE	UBYTE
35	23	io_Parm1	UBYTE	UBYTE
36	24	io_Parm2	UBYTE	UBYTE
37	25	io_Parm3	UBYTE	UBYTE

IORequest

32 bytes

C Include File: exec/io.h
ML Include File: exec/io.i

byte	hex	name	C type
0	00	io_Message	struct Message
20	14	io_Device	struct Device *
24	18	io_Unit	struct Unit *
28	1C	io_Command	UWORD
30	1E	io_Flags	UBYTE
31	1F	io_Error	BYTE

Note: There is no ML equivalent for this structure. Use IOStdReq (IO) instead.

IOStdReq

(IO in ML)
48 bytes

C Include File: exec/io.h
ML Include File: exec/io.i

byte	hex	name	C type	ML name	ML type
0	00	io_Message	struct Message		STRUCTURE
20	14	io_Device	struct Device *	IO_DEVICE	APTR
24	18	io_Unit	struct Unit *	IO_UNIT	APTR
28	1C	io_Command	UWORD	IO_COMMAND	UWORD
30	1E	io_Flags	UBYTE	IO_FLAGS	UBYTE
31	1F	io_Error	BYTE	IO_ERROR	BYTE
32	20	io_Actual	ULONG	IO_ACTUAL	ULONG
36	24	io_Length	ULONG	IO_LENGTH	ULONG
40	28	io_Data	APTR	IO_DATA	APTR
44	2C	io_Offset	ULONG	IO_OFFSET	ULONG

IOTArray

(TERMARRAY in ML)
8 bytes

C Include File: devices/serial.h
ML Include File: devices/serial.i

byte	hex	name	C type	ML name	ML type
0	00	TermArray0	ULONG	TERMARRAY_0	ULONG
4	04	TermArray1	ULONG	TERMARRAY_1	ULONG

IS

see Interrupt

Isrvstr

30 bytes
(New to 2.0)

C Include File: graphics/graphint.h
ML Include File: none

byte	hex	C name	C type
0	00	is_Node	struct Node
14	0E	Iptr	struct Isrvstr *
18	12	code	int(*)()
22	16	ccode	int(*)()
26	1A	Carg	int

IV

see IntVector

KeyMap

32 bytes

C Include File: devices/keymap.h
ML Include File: devices/keymap.i

byte	hex	name	C type	ML type
0	00	km_LoKeyMapTypes	UBYTE *	APTR
4	04	km_LoKeyMap	ULONG *	APTR
8	08	km_LoCapsable	UBYTE *	APTR
12	0C	km_LoRepeatable	UBYTE *	APTR
16	10	km_HiKeyMapTypes	UBYTE *	APTR
20	14	km_HiKeyMap	ULONG *	APTR
24	18	km_HiCapsable	UBYTE *	APTR
28	1C	km_HiRepeatable	UBYTE *	APTR

KeyMapNode

46 bytes

C Include File: devices/keymap.h
ML Include File: devices/keymap.i

byte	hex	name	C type	ML type
0	00	kn_Node	struct Node	STRUCT
14	0E	kn_KeyMap	struct KeyMap	STRUCT

KeyMapResource

28 bytes

C Include File: devices/keymap.h
ML Include File: devices/keymap.i

byte	hex	name	C type	ML type
0	00	kr_Node	struct Node	STRUCT
14	0E	kr_List	struct List	STRUCT

Layer

160 bytes
(Updated for 3.0)

C Include File: graphics/clip.h
ML Include File: graphics/clip.i

byte	hex	C name	C type	ML name	ML type
0	00	front	struct Layer *	lr_front	LONG
4	04	back	struct Layer *	lr_back	LONG
8	08	ClipRect	struct Cliprect *	lr_ClipRect	LONG
12	0C	rp	struct RastPort *	lr_rp	LONG
16	10	bounds	struct Rectangle	lr_MinX	WORD
18	12			lr_MinY	WORD
20	14			lr_MaxX	WORD
22	16			lr_MaxY	WORD
24	18	reserved	UBYTE[4]	lr_reserved	STRUCT
28	1C	priority	UWORD	lr_priority	WORD
30	1E	Flags	UWORD	lr_Flags	WORD
32	20	SuperBitMap	struct BitMap *	lr_SuperBitMap	LONG
36	24	SuperClipRect	struct ClipRect *	lr_SuperClipRect	LONG
40	28	Window	APTR	lr_Window	APTR
44	2C	ScrollX	WORD	lr_ScrollX	WORD
46	2E	ScrollY	WORD	lr_ScrollY	WORD
48	30	cr	struct ClipRect *	lr_cr	APTR
52	34	cr2	struct ClipRect *	lr_cr2	APTR
56	38	crnew	struct Cliprect *	lr_crnew	APTR
60	3C	SuperSaveClipRects	struct ClipRect *	lr_SuperSaveClipRects	APTR
64	40	cliprects	struct Cliprect *	lr_cliprects	APTR
68	44	LayerInfo	struct Layer_Info *	lr_LayerInfo	APTR
72	48	Lock	struct SignalSemaphore	lr_Lock	STRUCT
118	76	reserved3	UBYTE[8]	lr_reserved3	STRUCT
126	7E	ClipRegion	struct Region *	lr_ClipRegion	APTR
130	82	saveClipRects	struct Region *	lr_saveClipRects	APTR
134	86	Width	WORD	lr_reserved2	STRUCT
136	88	Height	WORD		
134	86	reserved2	UBYTE[18]		
156	9C	DamageList	struct Region *	lr_DamageList	APTR

Flags

name	bit	decimal	hex
LAYERSIMPLE	0	1	0001
LAYERSMART	1	2	0002
LAYERSUPER	2	4	0004
LAYERUPDATING	4	16	0010
LAYERBACKDROP	6	64	0040
LAYERREFRESH	7	128	0080
LAYER_CLIPRECTS_LOST	8	256	0100

Layer_Info

102 bytes
(Updated for 3.0)

C Include File: graphics/layers.h
ML Include File: graphics/layers.i

byte	hex	name	C type	ML type	ML type
0	00	top_layer	struct Layer *	li_top_layer	APTR
4	04	check_lp	struct Layer *	li_check_lp	APTR
8	08	obs	struct ClipRect *	li_obs	APTR
12	0C	FreeClipRects	struct ClipRect *	li_FreeClipRects	STRUCT
16	10	PrivateReserve1	LONG	li_PrivateReserve1	LONG
20	14	PrivateReserve2	LONG	li_PrivateReserve2	LONG
24	18	Lock	struct SignalSemaphore	li_Lock	STRUCT
70	46	gs_Head	struct MinList	li_gs_Head	STRUCT
82	52	PrivateReserve3	WORD	li_PrivateReserve3	WORD
84	54	PrivateReserve4	VOID *	li_PrivateReserve4	APTR
88	58	Flags	UWORD	li_Flags	WORD
90	5A	fatten_count	BYTE	li_fatten_count	BYTE
91	5B	LockLayersCount	BYTE	li_LockLayersCount	BYTE
92	5C	PrivateReserve5	WORD	li_PrivateReserve5	WORD
94	5E	BlankHook	VOID *	li_BlankHook	APTR
98	62	LayerInfo_extra	VOID *	li_LayerInfo_extra	APTR

Flags

name	bit	decimal	hex
LAYERSIMPLE	0	1	0001
LAYERSMART	1	2	0002
LAYERSUPER	2	4	0004
LAYERUPDATING	4	16	0010
LAYERBACKDROP	6	64	0040
LAYERREFRESH	7	128	0080

name	bit	decimal	hex
LAYER_CLIPRECTS_LOST	8	256	0100
LAYERIREFRESH	9	512	0200
LAYREIREFRESH2	10	1024	0400

LIB

see Library

Library

(LIB in ML)
34 bytes

C Include File: exec/libraries.h
ML Include File: exec/libraries.i

byte	hex	C name	C type	ML name	ML type
0	00	lib_Node	struct Node	LIB_NODE	STRUCTURE
14	0E	lib_Flags	UBYTE	LIB_FLAGS	UBYTE
15	0F	lib_pad	UBYTE	LIB_pad	UBYTE
16	10	lib_NegSize	UWORD	LIB_NEGSIZE	UWORD
18	12	lib_PosSize	UWORD	LIB_POSSIZE	UWORD
20	14	lib_Version	UWORD	LIB_VERSION	UWORD
22	16	lib_Revision	UWORD	LIB_REVISION	UWORD
24	18	lib_IdString	APTR	LIB_IDSTRING	APTR
28	22	lib_Sum	ULONG	LIB_SUM	ULONG
32	26	lib_OpenCnt	UWORD	LIB_OPENCNT	UWORD

lib_Flags

C flag name	ML bit name	bit	decimal	hex
LIBF_SUMMING	LIB,SUMMING	0	1	01
LIBF_CHANGED	LIB,CHANGED	1	2	02
LIBF_SUMUSED	LIB,SUMUSED	2	4	04
LIBF_DELEXP	LIB,DELEXP	3	8	08

Line

36 bytes
(New to 3.0)

C Include File: datatypes/textclass.h
ML Include File: datatypes/textclass.i

byte	hex	name	C type	ML type
0	00	ln_Link	struct MinNode	STRUCT
8	08	ln_Text	STRPTR	APTR
12	0C	ln_TextLen	ULONG	ULONG
16	10	ln_XOffset	UWORD	UWORD
18	12	ln_YOffset	UWORD	UWORD
20	14	ln_Width	UWORD	UWORD
22	16	ln_Height	UWORD	UWORD
24	18	ln_Flags	UWORD	UWORD
26	1A	ln_FgPen	BYTE	BYTE
27	1B	ln_BgPen	BYTE	BYTE
28	1C	ln_Style	ULONG	ULONG
32	20	ln_Data	APTR	APTR

ln_Flags

C flag name	ML bit name	bit	decimal	hex
LNF_LF	LN,LF	0	1	01
LNF_LINK	LN,LINK	1	2	02
LNF_OBJECT	LN,OBJECT	2	4	04
LNF_SELECTED	LN,SELECTED	3	8	08

List

14 bytes

C Include File: exec/lists.h
ML Include File: exec/lists.i

byte	hex	C name	C type	ML name	ML type
0	00	lh_Head	struct Node *	LH_HEAD	APTR
4	04	lh_Tail	struct Node *	LH_TAIL	APTR
8	08	lh_TailPred	struct Node *	LH_TAILPRED	APTR
12	0C	lh_Type	UBYTE	LH_TYPE	UBYTE
13	0D	lh_pad	UBYTE	LH_pad	UBYTE

lh_Type

name	bit	decimal	hex
NT_UNKNOWN	0	1	000001
NT_TASK	1	2	000002
NT_INTERRUPT	2	4	000004
NT_DEVICE	3	8	000008
NT_MSGPORT	4	16	000010
NT_MESSAGE	5	32	000020
NT_FREEMSG	6	64	000040
NT_REPLYMSG	7	128	000080

name	bit	decimal	hex
NT_RESOURCE	8	256	000100
NT_LIBRARY	9	512	000200
NT_MEMORY	10	1024	000400
NT_SOFTINT	11	2048	000800
NT_FONT	12	4096	001000
NT_PROCESS	13	8192	002000
NT_SEMAPHORE	14	16384	004000
NT_SIGNALSEM	15	32768	008000
NT_BOOTNODE	16	65536	010000

LoadSegBlock

512 bytes
(New to 2.0)

C Include File: devices/hardblocks.h
ML Include File: devices/hardblocks.i

byte	hex	name	C Type	ML type
0	00	lsb_ID	ULONG	ULONG
4	04	lsb_SummedLongs	ULONG	ULONG
8	08	lsb_ChkSum	LONG	LONG
12	0C	lsb_HostID	ULONG	ULONG
16	10	lsb_Next	ULONG	ULONG
20	14	lsb_LoadData	ULONG[123]	STRUCT

lsb_ID

name	decimal	hex
IDNAME_LOADSEG	1280525639	4C534547

LocalContextItem

20 bytes
(New to 2.0)

C Include File: libraries/iffpase.h
ML Include File: libraries/iffparse.i

byte	hex	name	C type	ML type
0	00	lci_Node	struct MinNode	STRUCTURE
8	08	lci_ID	ULONG	ULONG
12	0C	lci_Type	ULONG	ULONG
16	10	lci_Ident	ULONG	ULONG

Locale

168 bytes
(New to 3.0)

C Include File: libraries/locale.h
ML Include File: libraries/locale.i

byte	hex	name	C type	ML type
0	00	loc_LocaleName	STRPTR	APTR
4	04	loc_LanguageName	STRPTR	APTR
8	08	loc_PrefLanguages	STRPTR[10]	STRUCT
48	30	loc_Flags	ULONG	ULONG
52	34	loc_CodeSet	ULONG	ULONG
56	38	loc_CountryCode	ULONG	ULONG
60	3C	loc_TelephoneCode	ULONG	ULONG
64	40	loc_GMTOffset	LONG	LONG
68	44	loc_MeasuringSystem	UBYTE	UBYTE
69	45	loc_CalendarType	UBYTE	UBYTE
70	46	loc_Reserved0[2]	UBYTE	STRUCT
72	48	loc_DateTimeFormat	STRPTR	APTR
76	4C	loc_DateFormat	STRPTR	APTR
80	50	loc_TimeFormat	STRPTR	APTR
84	54	loc_ShortDateTimeFormat	STRPTR	APTR
88	58	loc_ShortDateFormat	STRPTR	APTR
92	5C	loc_ShortTimeFormat	STRPTR	APTR
96	60	loc_DecimalPoint	STRPTR	APTR
100	64	loc_GroupSeparator	STRPTR	APTR
104	68	loc_FracGroupSeparator	STRPTR	APTR
108	6C	loc_Grouping	UBYTE *	APTR
112	70	loc_FracGrouping	UBYTE *	APTR
116	74	loc_MonDecimalPoint	STRPTR	APTR
120	78	loc_MonGroupSeparator	STRPTR	APTR
124	7C	loc_MonFracGroupSeparator	STRPTR	APTR
128	80	loc_MonGrouping	UBYTE *	APTR
132	84	loc_MonFracGrouping	UBYTE *	APTR
136	88	loc_MonFracDigits	UBYTE	UBYTE
137	89	loc_MonIntFracDigits	UBYTE	UBYTE
138	8A	loc_Reserved1[2]	UBYTE	STRUCT
140	8C	loc_MonCS	STRPTR	APTR
144	90	loc_MonSmallCS	STRPTR	APTR
148	94	loc_MonIntCS	STRPTR	APTR
152	98	loc_MonPositiveSign	STRPTR	APTR

byte	hex	name	C type	ML type
156	9C	loc_MonPositiveSpaceSep	UBYTE	UBYTE
157	9D	loc_MonPositiveSignPos	UBYTE	UBYTE
158	9E	loc_MonPositiveCSPos	UBYTE	UBYTE
159	9F	loc_Reserved2	UBYTE	UBYTE
160	A0	loc_MonNegativeSign	STRPTR	APTR
164	A4	loc_MonNegativeSpaceSep	UBYTE	UBYTE
165	A5	loc_MonNegativeSignPos	UBYTE	UBYTE
166	A6	loc_MonNegativeCSPos	UBYTE	UBYTE
167	A7	loc_Reserved3	UBYTE	UBYTE

loc_MeasuringSystem

name	decimal	hex
MS_ISO	0	00
MS_AMERICAN	1	01
MS_IMPERIAL	2	02
MS_BRITISH	3	03

loc_CalendarType

name	decimal	hex
CT_7SUN	0	00
CT_7MON	1	01
CT_7TUE	2	02
CT_7WED	3	03
CT_7THU	4	04
CT_7FRI	5	05
CT_7SAT	6	06

loc_MonPositiveSpaceSep
loc_MonNegativeSpaceSep

name	decimal	hex
SS_NOSPACE	0	00
SS_SPACE	1	01

loc_MonPositiveSignPos
loc_MonNegativeSignPos

name	decimal	hex
SP_PARENS	0	00
SP_PREC_ALL	1	01
SP_SUCC_ALL	2	02
SP_PREC_CURR	3	03
SP_SUCC_CURR	4	04

loc_MonPositiveCSPos
loc_MonNegativeCSPos

name	decimal	hex
CSP_PRECEDES	0	00
CSP_SUCCEEDS	1	01

LocaleBase

36 bytes
(New to 3.0)

C Include File: libraries/locale.h
ML Include File: libraries/locale.i

byte	hex	name	C type	ML type
0	00	lb_LibNode	struct Library	STRUCTURE
34	22	lb_SysPatches	BOOL	BOOL

LocalePrefs

854 bytes
(New to 3.0)

C Include File: prefs/locale.h
ML Include File: prefs/locale.i

byte	hex	name	C type	ML type
0	0000	lp_Reserved	ULONG[4]	STRUCT
16	0010	lp_CountryName	char[32]	STRUCT
48	0030	lp_PreferredLanguages	char[10][30]	STRUCT
348	015C	lp_GMTOffset	LONG	LONG
352	0160	lp_Flags	ULONG	ULONG
356	0162	lp_CountryData	struct CountryPrefs	STRUCT

LocalVar

24 bytes
(New to 2.0)

C Include File: dos/var.h
ML Include File: dos/var.i

byte	hex	name	C type	ML type
0	00	lv_Node	struct Node	STRUCT
14	0E	lv_Flags	UWORD	UWORD
16	10	lv_Value	UBYTE *	APTR
20	14	lv_Len	ULONG	ULONG

LN—see Node

LVDrawMsg

24 bytes
(New to 3.0)

C Include File: libraries/gadtools.h
ML Include File: libraries/gadtools.i

byte	hex	name	C type	ML type
0	00	lvdm_MethodID	ULONG	ULONG
4	04	lvdm_RastPort	struct RastPort *	APTR
8	08	lvdm_DrawInfo	struct DrawInfo *	APTR
12	0C	lvdm_Rectangle	struct Rectangle	STRUCT
20	14	lvdm_State	ULONG	ULONG

lvdm_State

name	decimal	hex
LVR_NORMAL	0	00
LVR_SELECTED	1	01
LVR_NORMALDISABLED	2	02
LVR_SELECTEDDISABLED	8	08

MathIEEEBase

60 bytes
(New to 2.0)

C Include File: libraries/mathlibrary.h
ML Include File: libraries/mathlibrary.i

byte	hex	name	C type	ML type
0	00	MathIEEEBase_LibNode	struct Library	STRUCT
34	22	MathIEEEBase_reserved	unsigned char[18]	STRUCT
52	34	MathIEEEBase_TaskOpenLib	int(*)()	APTR
56	38	MathIEEEBase_TaskCloseLib	int(*)()	APTR

MathIEEEResource

44 bytes
(Updated for 3.0)

C Include File: resources/mathresource.h
ML Include File: resources/mathresource.i

byte	hex	name	C type	ML type
0	00	MathIEEEResource_Node	struct Node	STRUCT
14	0E	MathIEEEResource_Flags	unsigned short	USHORT
16	10	MathIEEEResource_BaseAddr	unsigned short *	APTR
20	14	MathIEEEResource_DblBasInit	void(*)()	APTR
24	18	MathIEEEResource_DblTransInit	void(*)()	APTR
28	1C	MathIEEEResource_SglBasInit	void(*)()	APTR
32	20	MathIEEEResource_SglTransInit	void(*)()	APTR
36	24	MathIEEEResource_ExtBasInit	void(*)()	APTR
40	28	MathIEEEResource_ExtTransInit	void(*)()	APTR

MathIEEEResource_Flags

C flag name	ML bit name	bit	decimal	hex
MATHIEEERESOURCEF_DBLBAS	MATHIEEERESOURCE,DBLBAS	0	1	01
MATHIEEERESOURCEF_DBLTRANS	MATHIEEERESOURCE,DBLTRANS	1	2	02
MATHIEEERESOURCEF_SGLBAS	MATHIEEERESOURCE,SGLBAS	2	4	04
MATHIEEERESOURCEF_SGLTRANS	MATHIEEERESOURCE,SGLTRANS	3	8	08
MATHIEEERESOURCEF_EXTBAS	MATHIEEERESOURCE,EXTBAS	4	16	10
MATHIEEERESOURCEF_EXTTRANS	MATHIEEERESOURCE,EXTTRANS	5	32	20

MC

see MemChunk

ME

see MemEntry

MemChunk

(MC in ML)
8 bytes

C Include File: exec/memory.h
ML Include File: exec/memory.i

byte	hex	C name	C type	ML name	ML type
0	00	mc_Next	struct MemChunk *	MC_NEXT	APTR
4	04	mc_Bytes	ULONG	MC_BYTES	ULONG

MemEntry

(ME in ML)
8 bytes

C Include File: exec/memory.h
ML Include File: exec/memory.i

byte	hex	C name	C type	ML name	ML type
0	00	meu_Reqs	struct MemReq *	ME_REQS	APTR
4	04	me_Length	ULONG	ME_LENGTH	ULONG

Note: An alias for ME_REQS is ME_ADDR and meu_reqs is also known as meu_Addr (type APTR).

me_reqs

C name	ML name	bit	decimal	hex
MEMF_PUBLIC	MEM,PUBLIC	0	1	000001
MEMF_CHIP	MEM,CHIP	1	2	000002
MEMF_FAST	MEM,FAST	2	4	000004
MEMF_CLEAR	MEM,CLEAR	16	65536	010000
MEMF_LARGEST	MEM,LARGEST	17	131072	020000

MemHandlerData

12 bytes
(New to 3.0)

C Include File: exec/memory.h
ML Include File: exec/memory.i

byte	hex	name	type
0	00	memh_RequestSize	ULONG
4	04	memh_RequestFlags	ULONG
8	08	memh_Flags	ULONG

MemHeader

(MH in ML)
32 bytes

C Include File: exec/memory.h
ML Include File: exec/memory.i

byte	hex	C name	C type	ML name	ML type
0	00	mh_Node	struct Node	MH	STRUCTURE
14	0E	mh_Attributes	UWORD	MH_ATTRIBUTES	UWORD
16	10	mh_First	struct MemChunk *	MH_FIRST	APTR
20	14	mh_Lower	APTR	MH_LOWER	APTR
24	18	mh_Upper	APTR	MH_UPPER	APTR
28	1C	mh_Free	ULONG	MH_FREE	ULONG

mh_Attributes (MH_ATTRIBUTES in ML)

C flag name	ML bit name	bit	decimal	hex
MEMF_PUBLIC	MEM,PUBLIC	0	1	000001
MEMF_CHIP	MEM,CHIP	1	2	000002
MEMF_FAST	MEM,FAST	2	4	000004
MEMF_LOCAL	MEM,LOCAL	8	256	000100
MEMF_24BITDMA	MEM,24BITDMA	9	512	000200
MEMF_CLEAR	MEM,CLEAR	16	65536	010000
MEMF_LARGEST	MEM,LARGEST	17	131072	020000
MEMF_REVERSE	MEM,REVERSE	18	262144	040000
MEMF_TOTAL	MEM,TOTAL	19	524288	080000

MemList

(ML in ML)
24 bytes

C Include File: exec/memory.h
ML Include File: exec/memory.i

byte	hex	C name	C type	ML name	ML type
0	00	ml_Node	struct Node	ML	STRUCTURE
14	0E	ml_NumEntries	UWORD	ML_NUMENTRIES	UWORD
16	10	ml_ME	struct MemEntry[1]	ML_ME	LABEL

Menu

30 bytes
(Updated for 3.0)

C Include File: intuition/intuition.h
ML Include File: intuition/intuition.i

byte	hex	C name	C type	ML name	ML type
0	00	NextMenu	struct Menu *	mu_NextMenu	APTR
4	04	LeftEdge	WORD	mu_LeftEdge	WORD
6	06	TopEdge	WORD	mu_TopEdge	WORD
8	08	Width	WORD	mu_Width	WORD
10	0A	Height	WORD	mu_Height	WORD
12	0C	Flags	UWORD	mu_Flags	WORD
14	0E	MenuName	BYTE *	mu_MenuName	APTR
18	12	FirstItem	struct MenuItem *	mu_FirstItem	APTR
22	16	JazzX	WORD	mu_JazzX	WORD
24	18	JazzY	WORD	mu_JazzY	WORD
26	1A	BeatX	WORD	mu_BeatX	WORD
28	1C	BeatY	WORD	mu_BeatY	WORD

Flags

name	bit	decimal	hex
MENUENABLED	0	1	0001
MIDRAWN	8	256	0100

MenuItem

34 bytes

C Include File: intuition/intuition.h
ML Include File: intuition/intuition.i

byte	hex	C name	C type	ML name	ML type
0	00	NextItem	struct MenuItem *	mi_NextItem	APTR
4	04	LeftEdge	WORD	mi_LeftEdge	WORD
6	06	TopEdge	WORD	mi_TopEdge	WORD
8	08	Width	WORD	mi_Width	WORD
10	0A	Height	WORD	mi_Height	WORD
12	0C	Flags	UWORD	mi_Flags	WORD
14	0E	MutualExclude	LONG	mi_MutualExclude	LONG
18	12	ItemFill	APTR	mi_ItemFill	APTR
22	16	SelectFill	APTR	mi_SelectFill	APTR

byte	hex	C name	C type	ML name	ML type
26	1A	Command	BYTE	mi_Command	BYTE
27	1B			mi_KludgeFill00	BYTE
28	1C	SubItem	struct MenuItem *	mi_SubItem	APTR
32	20	NextSelect	UWORD	mi_NextSelect	WORD

Flags

name	bit	decimal	hex
CHECKIT	0	1	0001
ITEMTEXT	1	2	0002
COMMSEQ	2	4	0004
MENUTOGGLE	3	8	0008
ITEMENABLED	4	16	0010
CHECKED	8	256	0100
ISDRAWN	12	4096	1000
HIGHITEM	13	8192	2000
MENUTOGGLED	14	16384	4000

Message

(MN in ML)
20 bytes

C Include File: exec/ports.h
ML Include File: exec/ports.i

byte	hex	name	C type	ML type
0	00	mn_Node	struct Node	STRUCT
14	0E	mn_ReplyPort	struct MsgPort *	APTR
18	12	mn_Length	UWORD	UWORD

MH

see MemHeader

MinList

(MLH in ML)
12 bytes

C Include File: exec/lists.h
ML Include File: exec/lists.i

byte	hex	C name	C type	ML name	ML type
0	00	mlh_Head	struct MinNode *	MLH_HEAD	APTR
4	04	mlh_Tail	struct MinNode *	MLH_TAIL	APTR
8	08	mlh_TailPred	struct MinNode *	MLH_TAILPRED	APTR

MinNode

(MLN in ML)
8 bytes

C Include File: exec/nodes.h
ML Include File: exec/nodes.i

byte	hex	C name	C type	ML name	ML type
0	00	mln_Succ	MinNode *	MLN_SUCC	APTR
4	04	mln_Pred	MinNode *	MLN_PRED	APTR

MiscResource

50 bytes
(Not defined in 2.0 or 3.0)

C Include File: resources/misc.h
ML Include File: resources/misc.i

byte	hex	name	C type	ML type
0	00	mr_Library	struct Library	STRUCT
34	22	mr_AllocArray	ULONG[4]	STRUCT

ML

see MemList

MLH

see MinList

MLN

see MinNode

MonitorInfo

96 bytes
(New to 2.0)

C Include File: graphics/displayinfo.h
ML Include File: graphics/displayinfo.i

byte	hex	C name	C type	ML name	ML type
0	00	Header	struct QueryHeader		
16	10	Mspc	struct MonitorSpec *	mtr_Mspc	APTR
20	14	ViewPosition	Point	mtr_ViewPosition	STRUCT
24	18	ViewResolution	Point	mtr_ViewResolution	STRUCT
28	1C	ViewPositionRange	struct Rectangle	mtr_ViewPositionRange	STRUCT
36	24	TotalRows	UWORD	mtr_TotalRows	UWORD
38	26	TotalColorClocks	UWORD	mtr_TotalColorClocks	UWORD
40	28	MinRow	UWORD	mtr_MinRow	UWORD
42	2A	Compatibility	WORD	mtr_Compatibility	WORD
44	2C	pad	UBYTE[36]	mtr_pad	STRUCT
80	50	DefaultViewPosition	Point	mtr_DefaultViewPosition	STRUCT
84	54	PreferredModeID	ULONG	mtr_PreferredModeID	ULONG
88	58	reserved	ULONG[2]	mtr_reserved	STRUCT

MonitorSpec

160 bytes
(Updated for 3.0)

C Include File: graphics/monitor.h
ML Include File: graphics/monitor.i

byte	hex	C name	C type	ML name	ML type
0	00	ms_Node	struct ExtendedNode		
24	18	ms_Flags	UWORD	ms_Flags	UWORD
26	1A	ratioh	LONG	ms_ratioh	LONG
30	1E	ratiov	LONG	ms_ratiov	LONG
34	22	total_rows	UWORD	ms_total_rows	UWORD
36	24	total_colorclocks	UWORD	ms_total_colorclocks	UWORD
38	26	DeniseMaxDisplayColumn	UWORD	ms_DeniseMaxDisplayColumn	UWORD
40	28	BeamCon0	UWORD	ms_BeamCon0	UWORD
42	2A	min_row	UWORD	ms_min_row	UWORD
44	2C	ms_Special	struct SpecialMonitor *	ms_Special	APTR
48	30	ms_OpenCount	UWORD	ms_OpenCount	UWORD
50	32	ms_transform	LONG(*)()	ms_transform	APTR
54	36	ms_translate	LONG(*)()	ms_translate	APTR
58	3A	ms_scale	LONG(*)()	ms_scale	APTR

byte	hex	C name	C type	ML name	ML type
62	3E	ms_xoffset	UWORD	ms_xoffset	UWORD
64	40	ms_yoffset	UWORD	ms_yoffset	UWORD
66	42	ms_LegalView	struct Rectangle	ms_LegalView	STRUCT
74	4A	ms_maxoscan	LONG(*)()	ms_maxoscan	APTR
78	4E	ms_videoscan	LONG(*)()	ms_videoscan	APTR
82	52	DeniseMinDisplayColumn	UWORD	ms_DeniseMinDisplayColumn	UWORD
84	54	DisplayCompatible	ULONG	ms_DisplayCompatible	UWORD
88	58	DisplayInfoDataBase	struct List	ms_DisplayInfoDataBase	STRUCT
102	66	DisplayInfoDataBaseSemaphore	struct SignalSemaphore	ms_DIDBSemaphore	STRUCT
148	94	ms_MrgCop	LONG(*)()	ms_MrgCop	ULONG
152	98	ms_LoadView	LONG(*)()	ms_LoadView	ULONG
156	9C	ms_KillView	LONG(*)()	ms_KillView	ULONG

mouth_rb

(MRB in ML)
92 bytes
(Updated for 2.0. Not defined in 3.0)

C Include File: devices/narrator.h
ML Include File: devices/narrator.i

byte	hex	C name	C type	ML name	ML type
0	00	voice	struct narrator_rb	MRB_VOICE	STRUCT
88	58	width	UBYTE	MRB_WIDTH	UBYTE
89	59	height	UBYTE	MRB_HEIGHT	UBYTE
90	5A	shape	UBYTE	MRB_SHAPE	UBYTE
91	5B	sync	UBYTE	MRB_SYNC	UBYTE

Msg

4 bytes
(New to 3.0)

C Include File: intuition/classusr.h
ML Include File: intuition/classusr.i

byte	hex	C name	C type	ML name	ML type
0	00	MethodID	ULONG	msg_MethodID	ULONG

MsgPort

(MP in ML)
34 bytes
(Updated for 3.0)

C Include File: exec/ports.h
ML Include File: exec/ports.i

byte	hex	C name	C type	ML name	ML type
0	00	mp_Node	struct Node	MP	STRUCTURE
14	0E	mp_Flags	UBYTE	MP_FLAGS	UBYTE
15	0F	mp_SigBit	UBYTE	MP_SIGBIT	UBYTE
16	10	mp_SigTask	void *	MP_SIGTASK	APTR
20	14	mp_MsgList	struct List	MP_MSGLIST	STRUCT

MN

see Message

MP

see MsgPort

MRB

see mouth_rb

NamedObject

4 bytes
(New to 3.0)
C Include File: utility/name.h
ML Include File: utility/name.i

byte	hex	name	C type	ML type
0	00	no_Object	VOID *	APTR

NameInfo

56 bytes
(New to 2.0)

C Include File: graphics/displayinfo.h
ML Include File: graphics/displayinfo.i

byte	hex	C name	C type	ML name	ML type
0	00	Header	struct QueryHeader	NameInfo	STRUCTURE
16	10	Name	UBYTE [32]	nif_Name	STRUCT
48	30	reserved	ULONG [2]	nif_reserved	STRUCT

narrator_rb

(NDI in ML)
88 bytes
(Changed for 2.0)

C Include File: devices/narrator.h
ML Include File: devices/narrator.i

byte	hex	C name	C type	ML name	ML type
0	00	message	struct	IOStdReq	
48	30	rate	UWORD	NDI_RATE	UWORD
50	32	pitch	UWORD	NDI_PITCH	UWORD
52	34	mode	UWORD	NDI_MODE	UWORD
54	36	sex	UWORD	NDI_SEX	UWORD
56	38	ch_masks	UBYTE *	NDI_CH_MASKS	APTR
60	3C	nm_masks	UWORD	NDI_NM_MASKS	UWORD
62	3E	volume	UWORD	NDI_VOLUME	UWORD
64	40	sampfreq	UWORD	NDI_SAMPFREQ	UWORD
66	42	mouths	UBYTE	NDI_MOUTHS	UBYTE
67	43	chanmask	UBYTE	NDI_CHANMASK	UBYTE
68	44	numchan	UBYTE	NDI_NUMCHAN	UBYTE
69	45	flags	UBYTE	NDI_FLAGS	UBYTE
70	46	F0enthusiasm	UBYTE	NDI_F0ENTHUSIASM	UBYTE
71	47	F0perturb	UBYTE	NDI_F0PERTURB	BYTE
72	48	F1adj	BYTE	NDI_F1ADJ	BYTE
73	49	F2adj	BYTE	NDI_F2ADJ	BYTE
74	4A	F3adj	BYTE	NDI_F3ADJ	BYTE
75	4B	A1adj	BYTE	NDI_A1ADJ	BYTE
76	4C	A2adj	BYTE	NDI_A2ADJ	BYTE
77	4D	A3adj	BYTE	NDI_A3ADJ	BYTE
78	4E	articulate	UBYTE	NDI_ARTICULATE	UBYTE
79	4F	centralize	UBYTE	NDI_CENTRALIZE	UBYTE
80	50	centphon	char *	NDI_CENTPHON	APTR
84	54	AVbias	BYTE	NDI_AVBIAS	BYTE
85	55	AFbias	BYTE	NDI_AFBIAS	BYTE
86	56	priority	BYTE	NDI_PRIORITY	BYTE
87	57	pad1	BYTE	NDI_PAD1	BYTE

Note: This structure is currently not found in 3.0.

NDI

see narrator_rb

NewAmigaGuide

52 bytes
(New to 3.0)

C Include File: libraries/amigaguide.h
ML Include File: libraries/amigaguide.i

byte	hex	name	C type	ML type
0	00	nag_Lock	BPTR	BPTR
4	04	nag_Name	STRPTR	APTR
8	08	nag_Screen	struct Screen *	APTR
12	0C	nag_PubScreen	STRPTR	APTR
16	10	nag_HostPort	STRPTR	APTR
20	14	nag_ClientPort	STRPTR	APTR
24	18	nag_BaseName	STRPTR	APTR
28	1C	nag_Flags	ULONG	ULONG
32	20	nag_Context	STRPTR *	APTR
36	24	nag_Node	STRPTR	APTR
40	28	nag_Line	LONG	LONG
44	2C	nag_Extens	struct TagItem *	APTR
48	30	nag_Client	VOID *	APTR

nag_Flags

C flag name	ML bit name	bit	decimal	hex
HTF_LOAD_INDEX	HT,LOAD_INDEX	0	1	000001
HTF_LOAD_ALL	HT,LOAD_ALL	1	2	000002
HTF_CACHE_NODE	HT,CACHE_NODE	2	4	000004
HTF_CACHE_DB	HT,CACHE_DB	3	8	000008
HTF_UNIQUE	HT,UNIQUE	15	32768	008000
HTF_NOACTIVATE	HT,NOACIVATE	16	65536	010000

NewBroker

26 bytes
(Updated for 3.0)

C Include File: libraries/commodities.h
ML Include File: libraries/commodities.i

byte	hex	C name	C type	ML name	ML type
0	00	nb_Version	BYTE	nb_Version	BYTE
1	01			nb_Reserve1	BYTE
2	02	nb_Name	BYTE *	nb_Name	APTR
6	06	nb_Title	BYTE *	nb_Title	APTR
10	0A	nb_Descr	BYTE *	nb_Descr	APTR
14	0E	nb_Unique	WORD	nb_Unique	WORD
16	10	nb_Flags	WORD	nb_Flags	WORD
18	12	nb_Pri	BYTE	nb_Pri	BYTE
19	13			nb_Reserve2	BYTE
20	14	nb_Port	struct MsgPort *	nb_Port	APTR
24	18	nb_ReservedChannel	WORD	nb_ReservedChannel	WORD

nb_Unique

name	decimal	hex
NBU_DUPLICATE	0	00
NBU_UNIQUE	1	01
NBU_NOTIFY	2	02

nb_Flags

name	decimal	hex
COF_SHOW_HIDE	4	04

NewGadget

30 bytes
(New to 2.0)

C Include File: libraries/gadtools.h
ML Include File: libraries/gadtools.i

byte	hex	C name	C type	ML name	ML type
0	00	ng_LeftEdge	WORD	gng_LeftEdge	WORD
2	02	ng_TopEdge	WORD	gng_TopEdge	WORD
4	04	ng_Width	WORD	gng_Width	WORD
6	06	ng_Height	WORD	gng_Height	WORD
8	08	ng_GadgetText	UBYTE *	gng_GadgetText	APTR
12	0C	ng_TextAttr	struct TextAttr *	gng_TextAttr	APTR
16	10	ng_GadgetID	UWORD	gng_GadgetID	UWORD
18	12	ng_Flags	ULONG	gng_Flags	ULONG
22	16	ng_VisualInfo	APTR	gng_VisulaInfo	APTR
26	1A	ng_UserData	APTR	gng_UserData	APTR

ng_Flags (gng_Flags in ML)

name	bit	decimal	hex
PLACETEXT_LEFT	0	1	01
PLACETEXT_RIGHT	1	2	02
PLACETEXT_ABOVE	2	4	04
PLACETEXT_BELOW	3	8	08
PLACETEXT_IN	4	16	10
NG_HIGHLABEL	5	32	20

NewMenu

20 bytes
(New to 2.0)

C Include File: libraries/gadtools.h
ML Include File: libraries/gadtools.i

byte	hex	C name	C type	ML name	ML type
0	00	nm_Type	UBYTE	nm_Type	UBYTE
1	01			nm_Pad	UBYTE
2	02	nm_Label	STRPTR	nm_Label	APTR
6	06	nm_CommKey	STRPTR	nm_CommKey	APTR
10	0A	nm_Flags	UWORD	nm_Flags	UWORD
12	0C	nm_MutualExclude	LONG	nm_MutualExclude	LONG
16	10	nm_UserData	APTR	nm_UserData	APTR

nm_Type

name	decimal	hex
NM_END	0	00
NM_TITLE	1	01
NM_ITEM	2	02
NM_SUB	3	03
NM_IGNORE	64	40
IM_ITEM	130	82
IM_SUB	131	83

NewScreen

32 bytes
(Updated for 3.0)

C Include File: intuition/screens.h
ML Include File: intuition/screens.i

byte	hex	C name	C type	ML name	ML type
0	00	LeftEdge	WORD	ns_LeftEdge	WORD
2	02	TopEdge	WORD	ns_TopEdge	WORD
4	04	Width	WORD	ns_Width	WORD
6	06	Height	WORD	ns_Height	WORD
8	08	Depth	WORD	ns_Depth	WORD
10	0A	DetailPen	UBYTE	ns_DetailPen	BYTE
11	0B	BlockPen	UBYTE	ns_BlockPen	BYTE
12	0C	ViewModes	UWORD	ns_ViewModes	WORD
14	0E	Type	UWORD	ns_Type	WORD
16	10	Font	struct TextAttr *	ns_Font	APTR
20	14	DefaultTitle	UBYTE *	ns_DefaultTitle	APTR
24	18	Gadgets	struct Gadget *	ns_Gadgets	APTR
28	1C	CustomBitMap	BitMap *	ns_CustomBitMap	APTR

ViewModes (ns_ViewModes in ML)

C name	ML name	bit	decimal	hex
GENLOCK_VIDEO	GENLOCK VIDEO	1	2	0002
LACE	V_LACE	2	4	0004
PFBA	V_PFBA	6	64	0040
EXTRA_HALFBRITE	7	128	0080	
GENLOCK_AUDIO	8	256	0100	
DUALPF	V_DUALPF	10	1024	0400
HAM	V_HAM	11	2048	0800
VP_HIDE		13	8192	2000
SPRITES	V_SPRITES	14	16384	4000
HIRES	V_HIRES	15	32768	8000

Type (ns_Type in ML)

name	bit	decimal	hex
SHOWTITLE	4	16	0010
BEEPING	5	32	0020
CUSTOMBITMAP	6	64	0040
SCREENBEHIND	7	128	0080
SCREENQUIET	8	256	0100

NewWindow

48 bytes
(Updated for 3.0)

C Include File: intuition/intuition.h
ML Include File: intuition/intuition.i

byte	hex	C name	C type	ML name	ML type
0	00	LeftEdge	WORD	nw_LeftEdge	WORD
2	02	TopEdge	WORD	nw_TopEdge	WORD
4	04	Width	WORD	nw_Width	WORD
6	06	Height	WORD	nw_Height	WORD
8	08	DetailPen	UBYTE	nw_DetailPen	BYTE
9	09	BlockPen	UBYTE	nw_BlockPen	BYTE
10	0A	IDCMPFlags	ULONG	nw_IDCMPFlags	LONG
14	0E	Flags	ULONG	nw_Flags	LONG
18	12	FirstGadget	struct Gadget *	nw_FirstGadget	APTR
22	16	CheckMark	struct Image *	nw_CheckMark	APTR
26	1A	Title	UBYTE *	nw_Title	APTR
30	1E	Screen	struct Screen *	nw_Screen	APTR
34	22	BitMap	struct BitMap *	nw_BitMap	APTR
38	26	MinWidth	WORD	nw_MinWidth	WORD
40	28	MinHeight	WORD	nw_MinHeight	WORD
42	2A	MaxWidth	UWORD	nw_MaxWidth	WORD
44	2C	MaxHeight	UWORD	nw_MaxHeight	WORD
46	2E	Type	UWORD	nw_Type	WORD

IDCMPFlags

name	bit	decimal	hex
SIZEVERIFY	0	1	00000001
NEWSIZE	1	2	00000002
REFRESHWINDOW	2	4	00000004
MOUSEBUTTONS	3	8	00000008
MOUSEMOVE	4	16	00000010
GADGETDOWN	5	32	00000020
GADGETUP	6	64	00000040
REQSET	7	128	00000080
MENUPICK	8	256	00000100
CLOSEWINDOW	9	512	00000200
RAWKEY	10	1024	00000400
REQVERIFY	11	2048	00000800
REQCLEAR	12	4096	00001000
MENUVERIFY	13	8192	00002000
NEWPREFS	14	16384	00004000
DISKINSERTED	15	32768	00008000
DISKREMOVED	16	65536	00010000
WBENCHMESSAGE	17	131072	00020000
ACTIVEWINDOW	18	262144	00040000
INACTIVEWINDOW	19	524288	00080000
DELTAMOVE	20	1048576	00100000
VANILLAKEY	21	2097152	00200000
INTUITICKS	22	4194304	00400000
LONELYMESSAGE	31	2147483648	80000000

Flags

name	bit	decima l	hex
WINDOWSIZING	0	1	00000001
WINDOWDRAG	1	2	00000002
WINDOWDEPTH	2	4	00000004
WINDOWCLOSE	3	8	00000008
SIZEBRIGHT	4	16	00000010
SIZEBBOTTOM	5	32	00000020
BACKDROP	8	256	00000100
REPORTMOUSE	9	512	00000200
GIMMEZEROZERO	10	1024	00000400
BORDERLESS	11	2048	00000800
ACTIVATE	12	4096	00001000
WINDOWACTIVE	13	8192	00002000
INREQUEST	14	16384	00004000
MENUSTATE	15	32768	00008000
RMBTRAP	16	65536	00010000
NOCAREREFRESH	17	131072	00020000
WINDOWREFRESH	24	16777216	01000000
WBENCHWINDOW	25	33554432	02000000
WINDOWTICKED	26	67108864	04000000

Type

name	bit	decimal	hex
SHOWTITLE	4	16	0010
BEEPING	5	32	0020
CUSTOMBITMAP	6	64	0040
SCREENBEHIND	7	128	0080
SCREENQUIET	8	256	0100

NexxStr

16 bytes
(New to 2.0)

C Include File: rexx/storage.h
ML Include File: rexx/storage.i

byte	hex	name	C type	ML type
0	00	ns_lvalue	LONG	LONG
4	04	ns_Length	UWORD	UWORD
6	06	ns_Flags	UBYTE	UBYTE
7	07	ns_Hash	UBYTE	UBYTE
8	08	ns_Buff	BYTE[8]	STRUCT

Node

(LN in ML)
14 bytes

C Include File: exec/nodes.h
ML Include File: exec/nodes.i

byte	hex	C name	C type	ML name	ML type
0	00	ln_Succ	struct Node *	LN_SUCC	APTR
4	04	ln_Pred	struct Node *	LN_PRED	APTR
8	08	ln_Type	UBYTE	LN_TYPE	UBYTE
9	09	ln_Pri	BYTE	LN_PRI	BYTE
10	0A	ln_Name	char *	LN_NAME	APTR

ln_Type

name	decimal	hex
NT_UNKNOWN	0	00
NT_TASK	1	01
NT_INTERRUPT	2	02
NT_DEVICE	3	03
NT_MSGPORT	4	04
NT_MESSAGE	5	05
NT_FREEMSG	6	06
NT_REPLYMSG	7	07
NT_RESOURCE	8	08
NT_LIBRARY	9	09
NT_MEMORY	10	0A
NT_SOFTINT	11	0B
NT_FONT	12	0C
NT_PROCESS	13	0D
NT_SEMAPHORE	14	0E
NT_SIGNALSEM	15	0F
NT_BOOTNODE	16	10
NT_KICKMEM	17	11
NT_GRAPHICS	18	12
NT_DEATHMESSAGE	19	13
NT_USER	254	FE
NT_EXTENDED	255	FF

NotifyMessage

38 bytes
(New to 2.0)

C Include File: dos/notify.h
ML Include File: dos/notify.i

byte	hex	name	C type	ML type
0	00	nm_ExecMessage	struct Message	STRUCT
20	14	nm_Class	ULONG	ULONG
24	18	nm_Code	UWORD	UWORD
26	1A	nm_NReq	struct NotifyRequest *	APTR
30	1E	nm_DoNotTouch	ULONG	ULONG
34	22	nm_DoNotTouch2	ULONG	ULONG

NotifyRequest

48 bytes
(New to 2.0)

C Include File: dos/notify.h
ML Include File: dos/notify.i

byte	hex	C name	C type	ML name	ML type
0	00	nr_Name	UBYTE *	nr_Name	CPTR
4	04	nr_FullName	UBYTE *	nr_FullName	CPTR
8	08	nr_UserData	ULONG	nr_UserData	ULONG
12	0C	nr_Flags	ULONG	nr_Flags	ULONG
16	10	nr_stuff.nr_Msg.nr_Port	struct MsgPort *		
16	10			nr_Port	APTR
16	10			nr_Task	APTR
16	10	nr_stuff.nr_Signal.nr_Task	struct Task *		
20	14	nr_stuff.nr_Signal.nr_SignalNum	UBYTE	nr_SignalNum UBYTE	
21	15	nr_stuff.nr_Signal.nr_pad	UBYTE[3]	nr_pad	STRUCT
24	18	nr_Reserved	ULONG[4]	nr_Reserved	STRUCT
40	28	nr_MsgCount	ULONG	nr_MsgCount	ULONG
44	2C	nr_Handler		struct MsgPort * nr_Handler	APTR

_Object

12 bytes
(New to 2.0)

C Include File: intuition/classes.h
ML Include File: intuition/classes.i

byte	hex	name	C type	ML type
0	00	o_Node	struct MinNode	STRUCT
8	08	o_Class	struct IClass *	APTR

OldDrawerData

56 bytes
(New to 2.0)

C Include File: workbench/workbench.h

byte	hex	C name	C type
0	00	dd_NewWindow	struct NewWindow
48	30	dd_CurrentX	LONG
52	34	dd_CurrentY	LONG

Note: In machine language, use the DrawerData structure.

opAddTail

8 bytes
(New to 2.0)

C Include File: intuition/classusr.h
ML Include File: intuition/classusr.i

byte	hex	C name	C type	ML name	ML type
0	00	MethodID	ULONG		
4	04	opat_List	struct List *	opta_List	APTR

opExpungeNode

8 bytes
(New to 3.0)

C Include File: libraries/amigaguide.h
ML Include File: libraries/amigaguide.i

byte	hex	name	C type	ML type
0	00	MethodID	ULONG	
4	04	oen_Attrs	struct TagItem *	APTR

opGet

12 bytes
(New to 2.0)

C Include File: intuition/classusr.h
ML Include File: intuition/classusr.i

byte	hex	C name	C type	ML name	ML type
0	00	MethodID	ULONG		
4	04	opg_AttrID	ULONG	opg_AttrID	ULONG
8	08	opg_Storage	ULONG *	opg_Storage	APTR

opFindHost

28 bytes
(New to 3.0)

C Include File: libraries/amigaguide.h
ML Include File: libraries/amigaguide.i

byte	hex	name	C type	ML type
0	00	MethodID	ULONG	
4	04	ofh_Attrs	struct TagItem *	APTR
8	08	ofh_Node	STRPTR	APTR
12	0C	ofn_TOC	STRPTR	APTR
16	10	ofn_Title	STRPTR	APTR
20	14	ofn_Next	STRPTR	APTR
24	18	ofn_Prev	STRPTR	APTR

opMember

8 bytes
(New to 2.0)

C Include File: intuition/classusr.h
ML Include File: intuition/classusr.i

byte	hex	C name	C type	ML name	ML type
0	00	MethodID	ULONG		
4	04	opam_Object	Object *	opam_Object	APTR

opNodeIO

28 bytes
(New to 3.0)

C Include File: libraries/amigaguide.h
ML Include File: libraries/amigaguide.i

byte	hex	name	C type	ML type
0	00	MethodID	ULONG	
4	04	onm_Attrs	struct TagItem *	APTR
8	08	onm_Node	STRPTR	APTR
12	0C	onm_FileName	STRPTR	APTR
16	10	onm_DocBuffer	STRPTR	APTR
20	14	onm_BuffLen	ULONG	ULONG
24	18	onm_Flag	ULONG	ULONG

opSet

12 bytes
(New to 2.0)
C Include: intuition/classusr.h
ML Include: intuition/classusr.i

byte	hex	C name	C type	ML name	ML type
0	00	MethodID	ULONG		
4	04	ops_AttrList	struct TagItem *	ops_AttrList	APTR
8	08	ops_GInfo	struct GadgetInfo *	ops_GInfo	APTR

opUpdate

16 bytes
(New to 2.0)

C Include File: intuition/classusr.h
ML Include File: intuition/classusr.i

byte	hex	C name	C type	ML name	ML type
0	00	MethodID	ULONG		
4	04	opu_AttrList	struct TagItem *	opu_AttrList	APTR
8	08	opu_GInfo	struct GadgetInfo *	opu_Gadget	APTR
12	0C	opu_Flags	ULONG	opu_Flags	ULONG

OverScanPrefs

36 bytes
(New to 3.0)

C Include File: prefs/overscan.h
ML Include File: prefs/overscan.i

byte	hex	name	C Type	ML type
0	00	os_Reserved	ULONG[4]	STRUCT
16	10	os_DisplayID	ULONG	ULONG
20	14	os_ViewPos	Point	STRUCT
24	18	os_Text	Point	STRUCT
28	1C	os_Standard	struct Rectangle	STRUCT

PaletteExtra

68 bytes
(New to 3.0)

C Include File: graphics/view.h
ML Include File: graphics/view.i

byte	hex	name	C type	ML type
0	00	pe_Semaphore	struct SignalSemaphore	STRUCT
46	2E	pe_FirstFree	UWORD	UWORD
48	30	pe_NFree	UWORD	UWORD
50	32	pe_FirstShared	UWORD	UWORD
52	34	pe_NShared	UWORD	UWORD

byte	hex	name	C type	ML type
54	36	pe_RefCnt	UBYTE *	APTR
58	3A	pe_AllocList	UBYTE *	APTR
62	3E	pe_ViewPort	struct ViewPort *	APTR
66	42	pe_SharableColors	UWORD	UWORD

PalettePrefs

384 bytes
(New to 3.0)

C Include File: prefs/palette.h
ML Include File: prefs/palette.i

byte	hex	name	C type	ML type
0	00	pap_Reserved	LONG[4]	STRUCT
16	10	pap_4ColorPens	UWORD[32]	STRUCT
80	50	pap_8ColorPens	UWORD[32]	STRUCT
128	80	pap_Colors	struct ColorSpec[32]	STRUCT

PartitionBlock

256 bytes
(New to 2.0)

C Include File: devices/hardblocks.h
ML Include File: devices/hardblocks.i

byte	hex	name	C type	ML type
0	00	pb_ID	ULONG	ULONG
4	04	pb_SummedLongs	ULONG	ULONG
8	08	pb_ChkSum	LONG	LONG
12	0C	pb_HostID	ULONG	ULONG
16	10	pb_Next	ULONG	ULONG
20	14	pb_Flags	ULONG	ULONG
24	18	pb_Reserved1	ULONG[2]	STRUCT
32	20	pb_DevFlags	ULONG	ULONG
36	24	pb_DriveName	UBYTE[32]	STRUCT
68	44	pb_Reserved2	ULONG[15]	STRUCT
128	80	pb_Environment	ULONG[17]	STRUCT
196	C4	pb_EReserved	ULONG[15]	STRUCT

pb_Flags

C flag name	C bit name	ML bit name	bit	decimal	hex
PBFF_BOOTABLE	PBFB_BOOTABLE	PBF,BOOTABLE	0	1	01
PBFB_NOMOUNT	PBFB_NOMOUNT	PBF,NOMOUNT	1	2	02

Point

4 bytes

C Include File: graphics/gfx.h
ML Include File: None

byte	hex	C name	C type
0	00	x	WORD
2	02	y	WORD

PointerPrefs

32 bytes
(New to 3.0)

C Include File: prefs/pointer.h
ML Include File: prefs/pointer.i

byte	hex	name	type
0	00	pp_Reserved	ULONG[4]
16	10	pp_Which	UWORD
18	12	pp_Size	UWORD
20	14	pp_Width	UWORD
22	16	pp_Height	UWORD
24	18	pp_Depth	UWORD
26	1A	pp_YSize	UWORD
28	1C	pp_X	UWORD
30	1E	pp_Y	UWORD

pp_Which

name	decimal	hex
WBP_NORMAL	0	00
WBP_BUSY	1	01

Preferences

232 bytes
(Updated for 3.0)

C Include File: intuition/preferences.h
ML Include File: intuition/preferences.i

byte	hex	C name	C type	ML name	ML type
0	00	FontHeight	BYTE	pf_FontHeight	BYTE
1	01	PrinterPort	UBYTE	pf_PrinterPort	BYTE
2	02	BaudRate	UWORD	pf_BaudRate	WORD
4	04	KeyRptSpeed	struct timeval	pf_KeyRptSpeed	STRUCT
12	0C	KeyRptDelay	struct timeval	pf_KeyRptDelay	STRUCT
20	14	DoubleClick	struct timeval	pf_DoubleClick	STRUCT
28	1C	PointerMatrix	UWORD[36]	pf_PointerMatrix	STRUCT
100	64	XOffset	BYTE	pf_XOffset	BYTE
101	65	YOffset	BYTE	pf_YOffset	BYTE
102	66	color17	UWORD	pf_color17	WORD
104	68	color18	UWORD	pf_color18	WORD
106	6A	color19	UWORD	pf_color19	WORD
108	6C	PointerTicks	UWORD	pf_PointerTicks	WORD
110	6E	color0	UWORD	pf_color0	WORD
112	70	color1	UWORD	pf_color1	WORD
114	72	color2	UWORD	pf_color2	WORD
116	74	color3	UWORD	pf_color3	WORD
118	76	ViewXOffset	BYTE	pf_ViewXOffset	BYTE
119	77	ViewYOffset	BYTE	pf_ViewYOffset	BYTE
120	78	ViewInitX	WORD	pf_ViewInitX	WORD
122	7A	ViewInitY	WORD	pf_ViewInitY	WORD
124	7C	EnableCLI	BOOL	pf_EnableCLI	BOOL
126	7E	PrinterType	UWORD	pf_PrinterType	WORD
128	80	PrinterFilename	UBYTE[30]	pf_PrinterFilename	STRUCT
158	9E	PrintPitch	UWORD	pf_PrintPitch	WORD
160	A0	PrintQuality	UWORD	pf_PrintQuality	WORD
162	A2	PrintSpacing	UWORD	pf_PrintSpacing	WORD
164	A4	PrintLeftMargin	UWORD	pf_PrintLeftMargin	WORD
166	A6	PrintRightMargin	UWORD	pf_PrintRightMargin	WORD
168	A8	PrintImage	UWORD	pf_PrintImage	WORD
170	AA	PrintAspect	UWORD	pf_PrintAspect	WORD
172	AC	PrintShade	UWORD	pf_PrintShade	WORD
174	AE	PrintThreshold	WORD	pf_PrintThreshold	WORD
176	B0	PaperSize	UWORD	pf_PaperSize	WORD
178	B2	PaperLength	UWORD	pf_PaperLength	WORD
180	B4	PaperType	UWORD	pf_PaperType	WORD

byte	hex	C name	C type	ML name	ML type
182	B6	SerRWBits	UBYTE	pf_SerRWBits	BYTE
183	B7	SerStopBuf	UBYTE	pf_SerStopBuf	BYTE
184	B8	SerParShk	UBYTE	pf_SerParShk	BYTE
185	B9	LaceWB	UBYTE	pf_LaceWB	BYTE
186	BA	WorkName	UBYTE[30]	pf_WorkName	STRUCT
216	D8	RowSizeChange	BYTE	pf_RowSizeChange	BYTE
217	D9	ColumnSizeChange	BYTE	pf_ColumnSizeChange	BYTE
218	DA	PrintFlags	UWORD	pf_PrintFlags	UWORD
220	DC	PrintMaxWidth	UWORD	pf_PrintMaxWidth	UWORD
222	DE	PrintMaxHeight	UWORD	pf_PrintMaxHeight	UWORD
224	E0	PrintDensity	UBYTE	pf_PrintDensity	UBYTE
225	E1	PrintXOffset	UBYTE	pf_PrintXOfset	UBYTE
226	E2	wb_Width	UWORD	pf_wb_Width	UWORD
228	E4	wb_Height	UWORD	pf_wb_Height	UWORD
230	E6	wb_Depth	UBYTE	pf_wb_Depth	UBYTE
231	E7	ext_size	UBYTE	pf_ext_size	UBYTE

PrinterPort (pf_PrinterPort in ML)

name	decimal	hex
PARALLEL_PRINTER	0	00
SERIAL_PRINTER	1	01

BaudRate (pf_BaudRate in ML)

name	decimal	hex
BAUD_110	0	00
BAUD_300	1	01
BAUD_1200	2	02
BAUD_2400	3	03
BAUD_4800	4	04
BAUD_9600	5	05
BAUD_19200	6	06
BAUD_MIDI	7	07

EnableCLI (pf_EnableCLI in ML)

name	decimal	hex
SCREEN_DRAG	16384	4000
MOUSE_ACCEL	32768	8000

PrinterType (pf_PrinterType in ML)

name	decimal	hex
CUSTOM_NAME	0	00
ALPHA_P_101	1	01
BROTHER_15XL	2	02

name	decimal	hex
CBM_MPS1000	3	03
DIAB_630	4	04
DIAB_ADV_D25	5	05
DIAB_C_150	6	06
EPSON	7	07
EPSON_JX_80	8	08
OKIMATE_20	9	09
QUME_LP_20	10	0A
HP_LASERJET	11	0B
HP_LASERJET_PLUS	12	0C

PrintPitch (pf_PrintPitch in ML)

name	decimal	hex
PICA	0	0000
ELITE	1024	0400
FINE	2048	0800

PrintQuality (pf_PrintQuality in ML)

name	decimal	hex
DRAFT	0	0000
LETTER	256	0100

PrintSpacing (pf_PrintSpacing in ML)

name	decimal	hex
SIX_LPI	0	0000
EIGHT_LPI	512	0200

PrintImage (pf_PrintImage in ML)

name	decimal	hex
IMAGE_POSITIVE	0	00
IMAGE_NEGATIVE	1	01

PrintAspect (pf_PrintAspect in ML)

name	decimal	hex
ASPECT_HORIZ	0	00
ASPECT_VERT	1	01

PrintShade (pf_PrintShade in ML)

name	decimal	hex
SHADE_BW	0	00
SHADE_GREYSCALE	1	01
SHADE_COLOR	2	02

PaperSize (pf_PaperSize in ML)

name	decimal	hex
US_LETTER	0	00
US_LEGAL	16	10
N_TRACTOR	32	20
W_TRACTOR	48	30
CUSTOM	64	40
EURO_A0	80	50
EURO_A1	96	60
EURO_A2	112	70
EURO_A3	128	80
EURO_A4	144	90
EURO_A5	160	A0
EURO_A6	176	B0
EURO_A7	192	C0
EURO_A8	208	D0

PaperType (pf_PaperType in ML)

name	decimal	hex
FANFOLD	0	00
SINGLE	128	80

SerRWBits (pf_SerRWBits in ML)

name	decimal	hex
SWRITE_BITS	15	0F
SREAD_BITS	240	F0

SerStopBuf (pf_SerStopBuf in ML)

name	decimal	hex
SBUF_512	0	00
SBUF_1024	1	01
SBUF_2048	2	02
SBUF_4096	3	03
SBUF_8000	4	04
SBUF_16000	5	05
SBUFSIZE_BITS	15	0F
SSTOP_BITS	240	F0

SerParShk (pf_SerParShk in ML)

name	decimal	hex
SHSHAKE_BITS	15	F0
SPARITY_BITS	240	0F

PrintFlags

name	bit	decimal	hex
CORRECT_RED	0	1	0001
CORRECT_GREEN	1	2	0002
CORRECT_BLUE	2	4	0004
CENTER_IMAGE	3	8	0008
BOUNDED_DIMENSIONS	4	16	0010
ABSOLUTE_DIMENSIONS	5	32	0020
PIXEL_DIMENSIONS	6	64	0040
MULTIPLY_DIMENSIONS	7	128	0080
INTEGER_SCALING	8	256	0100
HALFTONE_DITHERING	9	512	0200
FLOYD_DITHERING	10	1024	0400
ANTI_ALIAS	11	2048	0800
GREY_SCALE2	12	4096	1000

PrefHeader

6 bytes
(New to 3.0)

C Include File: prefs/prefhdr.h
ML Include File: prefs/prefhdr.i

byte	hex	name	type
0	00	ph_Version	UBYTE
1	01	ph_Type	UBYTE
2	02	ph_Flags	ULONG

PrinterData

6818 bytes
(Updated for 2.0)

C Include File: devices/prtbase.h
ML Include File: devices/prtbase.i

byte	hex	name	C type	ML type
0	0000	pd_Device	struct DeviceData	STRUCTURE
52	0034	pd_Unit	struct MsgPort	STRUCT
86	0056	pd_PrinterSegment	BPTR	BPTR
90	005A	pd_PrinterType	UWORD	UWORD
92	005C	pd_SegmentData	struct PrinterSegment *	APTR
96	0060	pd_PrintBuf	UBYTE *	APTR
100	0064	pd_PWrite	int (*)()	APTR

byte	hex	name	C type	ML type
104	0068	pd_PBothReady	int (*)()	APTR
108	006C	pd_p0	struct IOExtPar	STRUCT
190	00BE	pd_p1	struct IOExtPar	STRUCT
272	0110	pd_TIOR	struct timerequest	STRUCT
312	0138	pd_IORPort	struct MsgPort	STRUCT
346	015A	pd_TC	struct Task	STRUCT
438	01B6	pd_OldStk	UBYTE[2048]	STRUCT
2486	09B6	pd_Flags	UBYTE	UBYTE
2487	09B7	pd_pad	UBYTE	UBYTE
2488	09B8	pd_Preferences	struct Preferences	STRUCT
2720	0AA0	pd_PWaitEnabled	UBYTE	UBYTE
2721	0AA1	pd_Flags1	UBYTE	UBYTE
2722	0AA2	pd_Stk	UBYTE[4096]	STRUCT

PrinterExtendedData

66 bytes
(Expanded for 2.0)

C Include File: devices/prtbase.h
ML Include File: devices/prtbase.i

byte	hex	name	C type	ML type
0	00	ped_PrinterName	char *	APTR
4	04	ped_Init	VOID (*)()	APTR
8	08	ped_Expunge	VOID (*)()	APTR
12	0C	ped_Open	int (*)()	APTR
16	10	ped_Close	VOID (*)()	APTR
20	14	ped_PrinterClass	UBYTE	UBYTE
21	15	ped_ColorClass	UBYTE	UBYTE
22	16	ped_MaxColumns	UBYTE	UBYTE
23	17	ped_NumCharSets	UBYTE	UBYTE
24	18	ped_NumRows	UWORD	UWORD
26	1A	ped_MaxXDots	ULONG	ULONG
30	1E	ped_MaxYDots	ULONG	ULONG
34	22	ped_XDotsInch	UWORD	UWORD
36	24	ped_YDotsInch	UWORD	UWORD
38	26	ped_Commands	char ***	APTR
42	2A	ped_DoSpecial	int (*)()	APTR
46	2E	ped_Render	int (*)()	APTR
50	32	ped_TimeoutSecs	LONG	LONG
54	36	ped_8BitChars	int (*)()	APTR
58	3A	ped_PrintMode	LONG	LONG
62	3E	ped_ConvFunc	int (*)()	APTR

PrinterGfxPrefs

38 bytes
(New to 3.0)

C Include File: prefs/printergfx.h
ML Include File: prefs/printergfx.i

byte	hex	name	C type	ML type
0	00	pg_Reserved	LONG[4]	STRUCT
16	10	pg_Aspect	UWORD	UWORD
18	12	pg_Shade	UWORD	UWORD
20	14	pg_Image	UWORD	UWORD
22	16	pg_Threshold	WORD	WORD
24	18	pg_ColorCorrect	UBYTE	UBYTE
25	19	pg_Dimensions	UBYTE	UBYTE
26	1A	pg_Dithering	UBYTE	UBYTE
28	1C	pg_GraphicFlags	UWORD	UWORD
30	1E	pg_PrintDensity	UBYTE	UBYTE
32	20	pg_PrintMaxWidth	UWORD	UWORD
34	22	pg_PrintMaxHeight	UWORD	UWORD
36	24	pg_PrintXOffset	UBYTE	UBYTE
37	25	pg_PrintYOffset	UBYTE	UBYTE

pg_Aspect

name	decimal	hex
PA_HORIZONTAL	0	00
PA_VERTICAL	1	01

pg_Shade

name	decimal	hex
PS_BW	0	00
PS_GREYSCALE	1	01
PS_COLOR	2	02
PS_GREY_SCALE2	3	03

pg_Image

name	decimal	hex
PI_POSITIVE	0	00
PI_NEGATIVE	1	01

pg_ColorCorrect

C bit name	C flag name	ML bit name	bit	decimal	hex
PCCB_RED	PCCF_RED	PCC,RED	0	1	01
PCCB_GREEN	PCCF_GREEN	PCC,GREEN	1	2	02
PCCB_BLUE	PCCF_BLUE	PCC,BLUE	2	4	04

pg_Dimensions

name	decimal	hex
PD_IGNORE	0	00
PD_BOUNDED	1	01
PD_ABSOLUTE	2	02
PD_PIXEL	3	03
PD_MULTIPLY	4	04

pg_Dithering

name	decimal	hex
PD_ORDERED	0	00
PD_HALFTONE	1	01
PD_FLOYD	2	02

pg_GraphicsFlags

C bit name	C flag name	ML bit name	bit	decimal	hex
PGFB_CENTER_IMAGE	PGFF_CENTER_IMAGE	PGF,CENTER_IMAGE	0	1	01
PGFB_INTEGER_SCALING	PGFF_INTEGER_SCALING	PGF,INTEGER_SCALING	1	2	02
PGFB_ANTI_ALIAS	PGFF_ANTI_ALIAS	PGF,ANTI_ALIAS	2	4	04

PrinterPSPrefs

124 bytes
(New to 3.0)

C Include File: prefs/printerps.h
ML Include File: prefs/printerps.i

byte	hex	name	C type	ML type
0	00	ps_Reserved	LONG[4]	STRUCT
16	10	ps_DriverMode	UBYTE	UBYTE
17	11	ps_PaperFormat	UBYTE	UBYTE
18	12	ps_Reserved1	UBYTE[2]	STRUCT
20	14	ps_Copies	LONG	LONG
24	18	ps_PaperWidth	LONG	LONG
28	1C	ps_PaperHeight	LONG	LONG
32	20	ps_HorizontalDPI	LONG	LONG
36	24	ps_VerticalDPI	LONG	LONG

byte	hex	name	C type	ML type
40	28	ps_Font	UBYTE	UBYTE
41	29	ps_Pitch	UBYTE	UBYTE
42	2A	ps_Orientation	UBYTE	UBYTE
43	2B	ps_Tab	UBYTE	UBYTE
44	2C	ps_Reserved2	UBYTE[8]	STRUCT
52	34	ps_LeftMargin	LONG	LONG
56	38	ps_RightMargin	LONG	LONG
60	3C	ps_TopMargin	LONG	LONG
64	40	ps_BottomMargin	LONG	LONG
68	44	ps_FontPointSize	LONG	LONG
72	48	ps_Leading	LONG	LONG
76	4C	ps_Reserved3	UBYTE[8]	STRUCT
84	54	ps_LeftEdge	LONG	LONG
88	58	ps_TopEdge	LONG	LONG
92	5C	ps_Width	LONG	LONG
96	60	ps_Height	LONG	LONG
100	64	ps_Image	UBYTE	UBYTE
101	65	ps_Shading	UBYTE	UBYTE
102	66	ps_Dithering	UBYTE	UBYTE
103	67	ps_Transparency	UBYTE	UBYTE
104	68	ps_Reserved4	UBYTE[8]	STRUCT
112	70	ps_Aspect	UBYTE	UBYTE
113	71	ps_ScalingType	UBYTE	UBYTE
114	72	ps_ScalingMath	UBYTE	UBYTE
115	73	ps_Centering	UBYTE	UBYTE
116	74	ps_Reserved5	UBYTE[8]	STRUCT

ps_DriverMode

name	decimal	hex
DM_POSTSCRIPT	0	00
DM_PASSTHROUGH	1	01

ps_PaperFormat

name	decimal	hex
PF_USLETTER	0	00
PF_USLEGAL	1	01
PF_A4	2	02
PF_CUSTOM	3	03

ps_Font

name	decimal	hex
FONT_COURIER	0	00
FONT_TIMES	1	01
FONT_HELVETICA	2	02
FONT_HELV_NARROW	3	03
FONT_AVANTGARDE	4	04
FONT_BOOKMAN	5	05
FONT_NEWCENT	6	06
FONT_PALATINO	7	07
FONT_ZAPFCHANCERY	8	08

ps_Pitch

name	decimal	hex
PITCH_NORMAL	0	00
PITCH_COMPRESSED	1	01
PITCH_EXPANDED	2	02

ps_Orientation

name	decimal	hex
ORIENT_PORTRAIT	0	00
ORIENT_LANDSCAPE	1	01

ps_Tab

name	decimal	hex
TAB_4	0	00
TAB_8	1	01
TAB_QUART	2	02
TAB_HALF	3	03
TAB_INCH	4	04

ps_Image

name	decimal	hex
IM_POSITIVE	0	00
IM_NEGATIVE	1	01

ps_Shading

name	decimal	hex
SHAD_BW	0	00
SHAD_GREYSCALE	1	01
SHAD_COLOR	2	02

ps_Dithering

name	decimal	hex
DITH_DEFAULT	0	00
DITH_DOTTY	1	01
DITH_VERT	2	02
DITH_HORIZ	3	03
DITH_DIAG	4	04

ps_Transparency

name	decimal	hex
TRANS_NONE	0	00
TRANS_WHITE	1	01
TRANS_COLOR0	2	02

ps_Aspect

name	decimal	hex
ASP_HORIZ	0	00
ASP_VERT	1	01

ps_ScalingType

name	decimal	hex
ST_ASPECT_ASIS	0	00
ST_ASPECT_WIDE	1	01
ST_ASPECT_TALL	2	02
ST_ASPECT_BOTH	3	03
ST_FITS_WIDE	4	04
ST_FITS_TALL	5	05
ST_FITS_BOTH	6	06

ps_ScalingMath

name	decimal	hex
SM_INTEGER	0	00
SM_FRACTIONAL	1	01

ps_Centering

name	decimal	hex
CENT_NONE	0	00
CENT_HORIZ	1	01
CENT_VERT	2	02
CENT_BOTH	3	03

PrinterSegment

78 bytes
(Expanded for 2.0)

C Include File: devices/prtbase.h
ML Include File: devices/prtbase.i

byte	hex	name	C type	ML type
0	00	ps_NextSegment	ULONG	ULONG
4	04	ps_runAlert	ULONG	ULONG
8	08	ps_Version	UWORD	UWORD
10	0A	ps_Revision	UWORD	UWORD
12	0C	ps_PED	struct PrinterExtendedData	LABEL

PrinterTxtPrefs

64 bytes
(New to 3.0)

C Include File: prefs/printertxt.h
ML Include File: prefs/printertxt.i

byte	hex	name	C type	ML type
0	00	pt_Reserved	LONG[4]	STRUCT
16	10	pt_Driver	UBYTE[DRIVERNAMESIZE]	STRUCT
46	2E	pt_Port	UBYTE	UBYTE
48	30	pt_PaperType	UWORD	UWORD
50	32	pt_PaperSize	UWORD	UWORD
52	34	pt_PaperLength	UWORD	UWORD
54	36	pt_Pitch	UWORD	UWORD
56	38	pt_Spacing	UWORD	UWORD
58	3A	pt_LeftMargin	UWORD	UWORD
60	3C	pt_RightMargin	UWORD	UWORD
62	3E	pt_Quality	UWORD	UWORD

pt_Port

name	decimal	hex
PP_PARALLEL	0	00
PP_SERIAL	1	01

pt_PaperType

name	decimal	hex
PT_FANFOLD	0	00
PT_SINGLE	1	01

pt_PaperSize

name	decimal	hex
PS_US_LETTER	0	00
PS_US_LEGAL	1	01
PS_N_TRACTOR	2	02
PS_W_TRACTOR	3	03
PS_CUSTOM	4	04
PS_EURO_A0	5	05
PS_EURO_A1	6	06
PS_EURO_A2	7	07
PS_EURO_A3	8	08
PS_EURO_A4	9	09
PS_EURO_A5	10	0A
PS_EURO_A6	11	0B
PS_EURO_A7	12	0C
PS_EURO_A8	13	0D

pt_PrintPitch

name	decimal	hex
PP_PICA	0	00
PP_ELITE	1	01
PP_FINE	2	02

pt_PrintSpacing

name	decimal	hex
PP_SIX_LPI	0	00
PP_EIGHT_LPI	1	01

pt_PrintQuality

name	decimal	hex
PQ_DRAFT	0	00
PQ_LETTER	1	01

PrinterUnitPrefs

56 bytes
(New to 3.0)

C Include File: prefs/printertxt.h
ML Include File: prefs/printertxt.i

byte	hex	name	C type	ML type
0	00	pu_Reserved	LONG[4]	LONG 4*4
16	10	pu_UnitNum	LONG	LONG
20	14	pu_OpenDeviceFlags	ULONG	ULONG
24	18	pu_OpenDeviceName	UBYTE[DEVICENAMESIZE]	STRUCT

Process

228 bytes
(Expanded for 2.0)

C Include File: dos/dosextens.h
ML Include File: dos/dosextens.i

byte	hex	name	C type	ML type
0	00	pr_Task	struct Task	STRUCT
92	5C	pr_MsgPort	struct MsgPort	STRUCT
126	7E	pr_Pad	WORD	WORD
128	80	pr_SegList	BPTR	BPTR
132	84	pr_StackSize	LONG	LONG
136	88	pr_GlobVec	APTR	APTR
140	8C	pr_TaskNum	LONG	LONG
144	90	pr_StackBase	BPTR	BPTR
148	94	pr_Result2	LONG	LONG
152	98	pr_CurrentDir	BPTR	BPTR
156	9C	pr_CIS	BPTR	BPTR
160	A0	pr_COS	BPTR	BPTR
164	A4	pr_ConsoleTask	APTR	APTR
168	A8	pr_FileSystemTask	APTR	APTR
172	AC	pr_CLI	BPTR	BPTR
176	B0	pr_ReturnAddr	APTR	APTR
180	B4	pr_PktWait	APTR	APTR
184	B8	pr_WindowPtr	APTR	APTR
188	BC	pr_HomeDir	BPTR	BPTR
192	C0	pr_Flags	LONG	LONG
196	C4	pr_ExitCode	void (*)()	APTR

byte	hex	name	C type	ML type
200	C8	pr_ExitData	LONG	LONG
204	CC	pr_Arguments	UBYTE *	APTR
208	D0	pr_LocalVars	struct MinList	STRUCT
220	DC	pr_ShellPrivate	ULONG	APTR
224	E0	pr_CES	BPTR	BPTR

Note: This structure was previously located in libraries/dosextens.h (C version) and libraries/dosextens.i (ML version).

pr_Flags

C flag name	C bit name	ML bit name	bit	decimal	hex
PRF_FREESEGLIST	PRB_FREESEGLIST	PR,FREESEGLIST	0	1	01
PRF_FREECURRDIR	PRB_FREECURRDIR	PR,FREECURRDIR	1	2	02
PRF_FREECLI	PRB_CLI	PR,CLI	2	4	04
PRF_CLOSEINPUT	PRB_CLOSEINPUT	PR,CLOSEINPUT	3	8	08
PRF_CLOSEOUTPUT	PRB_CLOSEOUTPUT	PR,CLOSEOUTPUT	4	16	10
PRF_FREEARGS	PRB_FREEARGS	PR,FREEARGS	5	32	20

PropInfo

22 bytes
(Updated for 3.0)

C Include File: intuition/intuition.h
ML Include File: intuition/intuition.i

byte	hex	C name	C type	ML name	ML type
0	00	Flags	UWORD	pi_Flags	WORD
2	02	HorizPot	UWORD	pi_HorizPot	WORD
4	04	VertPot	UWORD	pi_VertPot	WORD
6	06	HorizBody	UWORD	pi_HorizBody	WORD
8	08	VertBody	UWORD	pi_VertBody	WORD
10	0A	CWidth	UWORD	pi_CWidth	WORD
12	0C	CHeight	UWORD	pi_CHeight	WORD
14	0E	HPotRes	UWORD	pi_HPotRes	WORD
16	10	VPotRes	UWORD	pi_VPotRes	WORD
18	12	LeftBorder	UWORD	pi_LeftBorder	WORD
20	14	TopBorder	UWORD	pi_TopBorder	WORD

Flags

name	decimal	hex
AUTOKNOB	1	0001
FREEHORIZ	2	0002
FREEVERT	4	0004
PROPBORDERLESS	8	0008
PROPNEWLOOK	16	0010
KNOBHIT	256	0100

PrtInfo

114 bytes
(New to 2.0)

C Include File: devices/prtgfx.h
ML Include File: devices/prtgfx.i

byte	hex	name	C type	ML type
0	00	pi_render	int(*)()	APTR
4	04	pi_rp	struct RastPort *	APTR
8	08	pi_temprp	struct RastPort *	APTR
12	0C	pi_RowBuf	UWORD *	APTR
16	10	pi_HamBuf	UWORD *	APTR
20	14	pi_ColorMap	union colorentry *	APTR
24	18	pi_ColorInt	union colorentry *	APTR
28	1C	pi_HamInt	union colorentry *	APTR
32	20	pi_Dest1Int	union colorentry *	APTR
36	24	pi_Dest2Int	union colorentry *	APTR
40	28	pi_ScaleX	UWORD *	APTR
44	2C	pi_ScaleXAlt	UWORD *	APTR
48	30	pi_dmatrix	UBYTE *	APTR
52	34	pi_TopBuf	UWORD *	APTR
56	38	pi_BotBuf	UWORD *	APTR
60	3C	pi_RowBufSize	UWORD	UWORD
62	3E	pi_HamBufSize	UWORD	UWORD
64	40	pi_ColorMapSize	UWORD	UWORD
66	42	pi_ColorIntSize	UWORD	UWORD
68	44	pi_HamIntSize	UWORD	UWORD
70	46	pi_Dest1IntSize	UWORD	UWORD
72	48	pi_Dest2IntSize	UWORD	UWORD
74	4A	pi_ScaleXSize	UWORD	UWORD
76	4C	pi_ScaleXAltSize	UWORD	UWORD
78	4E	pi_PrefsFlag	UWORD	UWORD
80	50	pi_special	ULONG	ULONG
84	54	pi_xstart	UWORD	UWORD

byte	hex	name	C type	ML type
86	56	pi_ystart	UWORD	UWORD
88	58	pi_width	UWORD	UWORD
90	5A	pi_height	UWORD	UWORD
92	5C	pi_pc	ULONG	ULONG
96	60	pi_pr	ULONG	ULONG
100	64	pi_ymult	UWORD	UWORD
102	66	pi_ymod	UWORD	UWORD
104	68	pi_ety	WORD	UWORD
106	6A	pi_xpos	UWORD	UWORD
108	6C	pi_threshold	UWORD	UWORD
110	6E	pi_tempwidth	UWORD	UWORD
112	70	pi_flags	UWORD	UWORD

PTERMARRAY

see IOPArray

PubScreenNode

30 bytes
(New to 2.0)

C Include File: intuition/screens.h
ML Include File: intuition/screens.i

byte	hex	C name	C type	ML name	ML type
0	00	psn_Node	struct Node		
14	0E	psn_Screen	struct Screen *	psn_Screen	APTR
18	12	psn_Flags	UWORD	psn_Flags	UWORD
20	14	psn_Size	WORD	psn_Size	WORD
22	16	psn_VisitorCount	WORD	psn_VisitorCount	WORD
24	18	psn_SigTask	struct Task *	psn_SigTask	APTR
28	1C	psn_SigBit	UBYTE	psn_SigBit	UBYTE
29	1D		psn_Pad1	UBYTE	

psn_Flags

name	bit	decimal	hex
SHANGHAI	0	1	01
POPPUBSCREEN	1	2	02

QueryHeader

16 bytes
(New to 2.0)

C Include File: graphics/displayinfo.h
ML Include File: graphics/displayinfo.i

byte	hex	C name	C type	ML name	ML type
0	00	StructID	ULONG	qh_StructID	ULONG
4	04	DisplayID	ULONG	qh_DisplayID	ULONG
8	08	SkipID	ULONG	qh_SkipID	ULONG
12	0C	Length	ULONG	qh_Length	ULONG

RasInfo

12 bytes
(Updated for 3.0)

C Include File: graphics/view.h
ML Include File: graphics/view.i

byte	hex	C name	C type	ML name	ML type
0	00	Next	struct RasInfo *	ri_Next	APTR
4	04	BitMap	struct BitMap *	ri_BitMap	LONG
8	08	RxOffset	WORD	ri_RxOffset	WORD
10	0A	RyOffset	WORD	ri_RyOffset	WORD

RastPort

100 bytes
(updated for 3.0)

C Include File: graphics/rastport.h
ML Include File: graphics/rastport.i

byte	hex	C name	C type	ML name	ML type
0	00	Layer	struct Layer *	rp_Layer	LONG
4	04	BitMap	struct BitMap *	rp_BitMap	LONG
8	08	AreaPtrn	UWORD *	rp_AreaPtrn	LONG
12	0C	TmpRas	struct TmpRas *	rp_TmpRas	LONG
16	10	AreaInfo	struct AreaInfo *	rp_AreaInfo	LONG
20	14	GelsInfo	struct GelsInfo *	rp_GelsInfo	LONG
24	18	Mask	UBYTE	rp_Mask	BYTE

byte	hex	C name	C type	ML name	ML type
25	19	FgPen	BYTE	rp_FgPen	BYTE
26	1A	BgPen	BYTE	rp_BgPen	BYTE
27	1B	AOlPen	BYTE	rp_AOlPen	BYTE
28	1C	DrawMode	BYTE	rp_DrawMode	BYTE
29	1D	AreaPtSz	BYTE	rp_AreaPtSz	BYTE
30	1E	linpatcnt	BYTE	rp_linpatcnt	BYTE
31	1F	dummy	BYTE	rp_dummy	BYTE
32	20	Flags	UWORD	rp_Flags	WORD
34	22	LinePtrn	UWORD	rp_LinePtrn	WORD
36	24	cp_x	UWORD	rp_cp_x	WORD
38	26	cp_y	UWORD	rp_cp_y	WORD
40	28	minterms	UBYTE[8]	rp_minterms	STRUCT
48	30	PenWidth	WORD	rp_PenWidth	WORD
50	32	PenHeight	WORD	rp_PenHeight	WORD
52	34	Font	struct TextFont *	rp_Font	LONG
56	38	AlgoStyle	UBYTE	rp_AlgoStyle	BYTE
57	39	TxFlags	UBYTE	rp_TxFlags	BYTE
58	3A	TxHeight	UWORD	rp_TxHeight	WORD
60	3C	TxWidth	UWORD	rp_TxWidth	WORD
62	3E	TxBaseLine	UWORD	rp_TxBaseLine	WORD
64	40	TxSpacing	WORD	rp_TxSpacing	WORD
66	42	RP_User	APTR *	rp_RP_User	APTR
70	46	longreserved	ULONG[2]	rp_longreserved	STRUCT
78	4E	wordreserved	UWORD[7]	rp_wordreserved	STRUCT
92	5C	reserved	UBYTE[8]	rp_reserved	STRUCT

DrawMode

C name	ML name	bit	decimal	hex
JAM2	RP_JAM2	0	1	01
COMPLEMENT	RP_COMPLEMENT	1	2	02
INVERSVID	RP_INVERSVID	2	4	04

Flags

C name	ML name	bit	decimal	hex
FRST_DOT	RP,FRST_DOT	0	1	01
ONE_DOT	RP,ONE_DOT	1	2	02
DBUFFER	RP,DBUFFER	2	4	04
AREAOUTLINE	RP,AREAOUTLINE	3	8	08
NOCROSSFILL	RP,NOCROSSFILL	5	32	20

AlgoStyle

C name	ML name	bit	decimal	hex
FSF_UNDERLINED	FS,UNDERLINED	0	1	01
FSF_BOLD	FS,BOLD	1	2	02
FSF_ITALIC	FS,ITALIC	2	4	04
FSF_EXTENDED	FS,EXTENDED	3	8	08

TxFlags

C name	ML name	bit	decimal	hex
FPF_ROMFONT	FP,ROMFONT	0	1	01
FPF_DISKFONT	FP,DISKFONT	1	2	02
FPF_REVPATH	FP,REVPATH	2	4	04
FPF_TALLDOT	FP,TALLDOT	3	8	08
FPF_LONGDOT	FP,LONGDOT	4	16	10
FPF_PROPORTIONAL	FP,PROPORTIONAL	5	32	20
FPF_DESIGNED	FP,DESIGNED	6	64	40
FPF_REMOVED	FP,REMOVED	7	128	80

RDArgs

32 bytes
(New to 2.0)

C Include File: dos/rdargs.h
ML Include File: dos/rdargs.i

byte	hex	name	C type	ML type
0	00	RDA_Source	struct CSource	STRUCT
12	0C	RDA_DAList	LONG	APTR
16	10	RDA_Buffer	UBYTE *	LONG
20	14	RDA_BufSiz	LONG	LONG
24	18	RDA_ExtHelp	UBYTE *	APTR
28	1C	RDA_Flags	LONG	LONG

RDA_Flags

C flag name	C bit name	ML bit name	bit	decimal	hex
RDAF_STDIN	RDAB_STDIN	RDA,STDIN	0	1	01
RDAF_NOALLOC	RDAB_NOALLOC	RDA,NOALLOC	1	2	02
RDAF_NOPROMPT	RDAB_NOPROMPT	DRA,NOPROMPT	2	4	04

Rectangle

8 bytes

C Include File: graphics/gfx.h
ML Include File: graphics/gfx.i

byte	hex	C name	C type	ML name	ML type
0	00	MinX	WORD	ra_MinX	WORD
2	02	MinY	WORD	ra_MinY	WORD
4	04	MaxX	WORD	ra_MaxX	WORD
6	06	MaxY	WORD	ra_MaxY	WORD

RecordLock

16 bytes
(New to 2.0)

C Include File: dos/record.h
ML Include File: dos/record.i

byte	hex	name	C type	ML type
0	00	rec_FH	BPTR	BPTR
4	04	rec_Offset	ULONG	ULONG
8	08	rec_Length	ULONG	ULONG
12	0C	rec_Mode	ULONG	ULONG

Rect32

16 bytes
(New to 2.0)

C Include File: graphics/gfx.h
ML Include File: graphics/gfx.i

byte	hex	C name	C type	ML name	ML type
0	00	MinX	LONG	r32_MinX	LONG
4	04	MinY	LONG	r32_MinY	LONG
8	08	MaxX	LONG	r32_MaxX	LONG
12	0C	MaxY	LONG	r32_MaxY	LONG

Region

12 bytes

C Include File: graphics/regions.h
ML Include File: graphics/regions.i

byte	hex	C name	C type	ML name	ML type
0	00	bounds	Rectangle	rg_bounds	STRUCT
8	08	RegionRectangle	struct RegionRectangle *	rg_RegionRectangle	APTR

RegionRectangle

16 bytes

C Include File: graphics/regions.h
ML Include File: graphics/regions.i

byte	hex	C name	C type	ML name	ML type
0	00	Next	struct RegionRectangle *	rr_Next	APTR
4	04	Prev	struct RegionRectangle *	rr_Prev	APTR
8	08	bounds	struct Rectangle	rr_bounds	STRUCT

Remember

12 bytes

C Include File: intuition/intuition.h
ML Include File: intuition/intuition.i

byte	hex	C name	C type	ML name	ML type
0	00	NextRemember	struct Remember *	rm_NextRemember	APTR
4	04	RememberSize	ULONG	rm_RememberSize	LONG
8	08	Memory	UBYTE *	rm_Memory	APTR

Requester

112 bytes
(Updated for 3.0)

C Include File: intuition/intuition.h
ML Include File: inutition/intuition.i

byte	hex	C name	C type	ML name	ML type
0	00	OlderRequester	struct Requester *	rq_OlderRequester	APTR
4	04	LeftEdge	WORD	rq_LeftEdge	WORD
6	06	TopEdge	WORD	rq_TopEdge	WORD
8	08	Width	WORD	rq_Width	WORD
10	0A	Height	WORD	rq_Height	WORD
12	0C	RelLeft	WORD	rq_RelLeft	WORD
14	0E	RelTop	WORD	rq_RelTop	WORD
16	10	ReqGadget	struct Gadget *	rq_ReqGadget	APTR
20	14	ReqBorder	struct Border *	rq_ReqBorder	APTR
24	18	ReqText	struct IntuiText *	rq_ReqText	APTR
28	1C	Flags	UWORD	rq_Flags	WORD
30	1E	BackFill	UBYTE	rq_BackFill	UBYTE
31	1F			rq_KludgeFill00	BYTE
32	20	ReqLayer	stryct Layer *	rq_ReqLayer	APTR
36	24	ReqPad1	UBYTE[32]	rq_ReqPad1	STRUCT
68	44	ImageBMap	struct BitMap *	rq_ImageBMap	APTR
72	48	RWindow	struct Window *	rq_RWindow	APTR
76	4C	ReqImage	struct Image *	rq_ReqImage	APTR
80	50	ReqPad2	UBYTE[32]	rq_ReqPad2	STRUCT

Flags

name	bit	decimal	hex
POINTREL	0	1	0001
PREDRAWN	1	2	0002
NOISYREQ	2	4	0004
REQOFFWINDOW	11	2048	0800
REQACTIVE	12	4096	1000
SYSREQUEST	13	8192	2000
DEFERREFRESH	14	16384	4000

Resident

(RT in ML)
26 bytes

C Include File: exec/resident.h
ML Include File: exec/resident.i

byte	hex	C name	C type	ML name	ML type
0	00	rt_MatchWord	UWORD	RT_MATCHWORD	UWORD
2	02	rt_MatchTag	struct Resident *	RT_MATCHTAG	APTR
6	06	rt_EndSkip	APTR	RT_ENDSKIP	APTR
10	0A	rt_Flags	UBYTE	RT_FLAGS	UBYTE
11	0B	rt_Version	UBYTE	RT_VERSION	UBYTE

byte	hex	C name	C type	ML name	ML type
12	0C	rt_Type	UBYTE	RT_TYPE	UBYTE
13	0D	rt_Pri	BYTE	RT_PRI	BYTE
14	0E	rt_Name	char *	RT_NAME	APTR
18	12	rt_IdString	char *	RT_IDSTRING	APTR
22	16	rt_Init	APTR	RT_INIT	APTR

rt_Flags

C flag name	ML bit name	bit	decimal	hex
RTF_COLDSTART	RT,COLDSTART	0	1	01
RTF_SINGLETASK	RT,SINGLETASK	1	2	02
RTF_AFTERDOS	RT,AFTERDOS	2	4	04
RTF_AUTOINIT	RT,AUTOINIT	7	128	80

RexxArg

16 bytes
(New to 2.0)

C Include File: rexx/storage.h
ML Include File: rexx/storage.i

byte	hex	name	C type	ML type
0	00	ra_Size	LONG	LONG
4	04	ra_Length	UWORD	UWORD
6	06	ra_Flags	UBYTE	UBYTE
7	07	ra_Hash	UBYTE	UBYTE
8	08	ra_Buff	BYTE[8]	STRUCT

ra_Flags

flag name	bit name	bit	decimal	hex
NSF_KEEP	NSB_KEEP	0	1	01
NSF_STRING	NSB_STRING	1	2	02
NSF_NOTNUM	NSB_NOTNUM	2	4	04
NSF_NUMBER	NSB_NUMBER	3	8	08
NSF_BINARY	NSB_BINARY	4	16	10
NSF_FLOAT	NSB_FLOAT	5	32	20
NSF_EXT	NSB_EXT	6	64	40
NSF_SOURCE	NSB_SOURCE	7	128	80

RexxMsg

128 bytes
(New to 2.0)

C Include File: rexx/storage.h
ML Include File: rexx/storage.i

byte	hex	C name	C type	ML name	ML type
0	00	rm_Node	struct Message		
20	14	rm_TaskBlock	APTR	rm_TaskBlock	APTR
24	18	rm_LibBase	APTR	rm_LibBase	APTR
28	1C	rm_Action	LONG	rm_Action	LONG
32	20	rm_Result1	LONG	rm_Result1	LONG
36	24	rm_Result2	LONG	rm_Result2	LONG
40	28	rm_Args	STRPTR[16]	rm_Args	STRUCT
104	68	rm_PassPort	struct MsgPort *	rm_PassPort	APTR
108	6C	rm_CommAddr	STRPTR	rm_CommAddr	APTR
112	70	rm_FileExt	STRPTR	rm_FileExt	APTR
116	74	rm_Stdin	LONG	rm_Stdin	LONG
120	78	rm_Stdout	LONG	rm_Stdout	LONG
124	7C	rm_avail	LONG	rm_avail	LONG

RexxMsgPort

80 bytes
(New to 2.0)

C Include File: rexx/rexxio.h
ML Include File: rexx/rexxio.i

byte	hex	C name	C type	ML name	ML type
0	00	rmp_Node	struct RexxRsrc		
32	20	rmp_Port	struct MsgPort	rmp_Port	STRUCT
66	42	rmp_ReplyList	struct List	rmp_ReplyList	STRUCT

RexxRsrc

32 bytes
(New to 2.0)

C Include File: rexx/storage.h
ML Include File: rexx/storage.i

byte	hex	C name	C type	ML name	ML type
0	00	rr_Node	struct Node		
14	0E	rr_Func	WORD	rr_Func	WORD
16	10	rr_Base	APTR	rr_Base	APTR
20	14	rr_Size	LONG	rr_Size	LONG
24	18	rr_Arg1	LONG	rr_Arg1	LONG
28	1C	rr_Arg2	LONG	rr_Arg2	LONG

RexxTask

330 bytes
(New to 2.0)

C Include File: rexx/storage.h
ML Include File: rexx/storage.i

byte	hex	C name	C type	ML name	ML type
0	0000	rt_Global	BYTE[200]		
200	00C8	rt_MsgPort	struct MsgPort	rt_MsgPort	STRUCT
234	00EA	rt_Flags	UBYTE	rt_Flags	UBYTE
235	00EB	rt_SigBit	BYTE	rt_SigBit	BYTE
236	00EC	rt_ClientID	APTR	rt_ClientID	APTR
240	00F0	rt_MsgPkt	APTR	rt_MsgPkt	APTR
244	00F4	rt_TaskID	APTR	rt_TaskID	APTR
248	00F8	rt_RexxPort	APTR	rt_RexxPort	APTR
252	00FC	rt_ErrTrap	APTR	rt_ErrTrap	APTR
256	0100	rt_StackPtr	APTR	rt_StackPtr	APTR
260	0104	rt_Header1	struct List	rt_Header1	STRUCT
274	0112	rt_Header2	struct List	rt_Header2	STRUCT
288	0120	rt_Header3	struct List	rt_Header3	STRUCT
302	012E	rt_Header4	struct List	rt_Header4	STRUCT
316	013C	rt_Header5	struct List	rt_Header5	STRUCT

rt_Flags

bit name	bit	decimal	hex
RTFB_TRACE	0	1	01
RTFB_HALT	1	2	02
RTFB_SUSP	2	4	04
RTFB_TCUSE	3	8	08
RTFB_WAIT	6	64	40
RTFB_CLOSE	7	128	80

RGBTable

3 bytes
(New to 3.0)

C Include File: prefs/pointer.h
ML Include File: prefs/pointer.i

byte	hex	C name	C type	ML name	ML type
0	00	t_Red	UBYTE	rgbt_Red	UBYTE
1	01	t_Green	UBYTE	rgbt_Green	UBYTE
2	02	t_Blue	UBYTE	rgbt_Blue	UBYTE

RigidDiskBlock

252 bytes
(New to 2.0)

C Include File: devices/hardblocks.h
ML Include File: devices/hardblocks.i

byte	hex	name	C type	ML type
0	00	rdb_ID	ULONG	ULONG
4	04	rdb_SummedLongs	ULONG	ULONG
8	08	rdb_ChkSum	LONG	LONG
12	0C	rdb_HostID	ULONG	ULONG
16	10	rdb_BlockBytes	ULONG	ULONG
20	14	rdb_Flags	ULONG	ULONG
24	18	rdb_BadBlockList	ULONG	ULONG
28	1C	rdb_PartitionList	ULONG	ULONG
32	20	rdb_FileSysHeaderList	ULONG	ULONG
36	24	rdb_DriveInit	ULONG	ULONG
40	28	rdb_Reserved1	ULONG[6]	STRUCT
64	40	rdb_Cylinders	ULONG	ULONG
68	44	rdb_Sectors	ULONG	ULONG
72	48	rdb_Heads	ULONG	ULONG
76	4C	rdb_Interleave	ULONG	ULONG
80	50	rdb_Park	ULONG	ULONG
84	54	rdb_Reserved2	ULONG[3]	STRUCT
96	60	rdb_WritePreComp	ULONG	ULONG
100	64	rdb_ReducedWrite	ULONG	ULONG
104	68	rdb_StepRate	ULONG	ULONG
108	6C	rdb_Reserved3	ULONG[5]	STRUCT
128	80	rdb_RDBBlocksLo	ULONG	ULONG

byte	hex	name	C type	ML type
132	84	rdb_RDBBlocksHi	ULONG	ULONG
136	88	rdb_LoCylinder	ULONG	ULONG
140	8C	rdb_HiCylinder	ULONG	ULONG
144	90	rdb_CylBlocks	ULONG	ULONG
148	94	rdb_AutoParkSeconds	ULONG	ULONG
152	98	rdb_Reserved4	ULONG	STRUCT
156	9C	rdb_DiskVendor	char[8]	STRUCT
164	A4	rdb_DiskProduct	char[16]	STRUCT
180	B4	rdb_DiskRevision	char[4]	STRUCT
184	B8	rdb_ControllerVendor	char[8]	STRUCT
192	C0	rdb_ControllerProduct	char[16]	STRUCT
208	D0	rdb_ControllerRevision	char[4]	STRUCT
212	D4	rdb_Reserved5	ULONG[10]	STRUCT

rdb_Flags

C flag name	C bit name	ML bit name	bit	decimal	hex
RDBFF_LAST	RDBFB_LAST	RDBF,LAST	0	1	01
RDBFF_LASTLUN	RDBFB_LASTLUN	RDBF,LASTLUN	1	2	02
RDBFF_LASTTID	RDBFB_LASTTID	RDBF,LASTTID	2	4	04
RDBFF_NORESELECT	RDBFB_NORESELECT	RDBF,NORESELECT	3	8	08
RDBFF_DISKID	RDBFB_DISKID	RDBF,DISKID	4	16	10
RDBFF_CTRLRID	RDBFB_CTRLRID	RDBF,CTRLRID	5	32	20
RDBFB_SYNCH	RDBFB_SYNCH	RDBF,SYNCH	6	64	40

RootNode

56 bytes
(Expanded for 2.0)

C Include File: dos/dosextens.h
ML Include File: dos/dosextens.i

byte	hex	name	C type	ML type
0	00	rn_TaskArray	BPTR	BPTR
4	04	rn_ConsoleSegment	BPTR	BPTR
8	08	rn_Time	struct DateStamp	STRUCT
20	14	rn_RestartSeg	LONG	LONG
24	18	rn_Info	BPTR	BPTR
28	1C	rn_FileHandlerSegment	BPTR	BPTR
32	20	rn_CliList	struct MinList	STRUCT
44	2C	rn_BootProc	struct MsgPort *	APTR
48	30	rn_ShellSegment	BPTR	BPTR
52	34	rn_Flags	LONG	LONG

Note: This structure was previously found in libraries/dosextens.h (C version) and libraries/dosextens.i (ML version).

RT

see Resident

RxsLib

252 bytes
(New to 2.0)

C Include File: rexx/rxslib.h
ML Include File: rexx/rxslib.i

byte	hex	C name	C type	ML name	ML type
0	00	rl_Node	struct Library		
34	22	rl_Flags	UBYTE	rl_Flags	UBYTE
35	23	rl_Shadow	UBYTE	rl_Shadow	UBYTE
36	24	rl_SysBase	APTR	rl_SysBase	APTR
40	28	rl_DOSBase	APTR	rl_DOSBase	APTR
44	2C	rl_IeeeDPBase	APTR	rl_IeeeDPBase	APTR
48	30	rl_SegList	LONG	rl_SegList	LONG
52	34	rl_NIL	LONG	rl_NIL	LONG
56	38	rl_Chunk	LONG	rl_Chunk	LONG
60	3C	rl_MaxNest	LONG	rl_MaxNest	LONG
64	40	rl_NULL	struct NexxStr *	rl_NULL	APTR
68	44	rl_FALSE	struct NexxStr *	rl_FALSE	APTR
72	48	rl_TRUE	struct NexxStr *	rl_TRUE	APTR
76	4C	rl_REXX	struct NexxStr *	rl_REXX	APTR
80	50	rl_COMMAND	struct NexxStr *	rl_COMMAND	APTR
84	54	rl_STDIN	struct NexxStr *	rl_STDIN	APTR
88	58	rl_STDOUT	struct NexxStr *	rl_STDOUT	APTR
92	5C	rl_STDERR	struct NexxStr *	rl_STDERR	APTR
96	60	rl_Version	STRPTR	rl_Version	APTR
100	64	rl_TaskName	STRPTR	rl_TaskName	APTR
104	68	rl_TaskPri	LONG	rl_TaskPri	LONG
108	6C	rl_TaskSeg	LONG	rl_TaskSeg	LONG
112	70	rl_StackSize	LONG	rl_StackSize	LONG
116	74	rl_RexxDir	STRPTR	rl_RexxDir	APTR
120	78	rl_CTABLE	STRPTR	rl_CTABLE	APTR
124	7C	rl_Notice	STRPTR	rl_Notice	APTR
128	80	rl_RexxPort	struct MsgPort	rl_RexxPort	STRUCT
162	A2	rl_ReadLock	UWORD	rl_ReadLock	UWORD
164	A4	rl_TraceFH	LONG	rl_TraceFH	APTR

byte	hex	C name	C type	ML name	ML type
168	A8	rl_TaskList	struct List	rl_TaskList	STRUCT
182	B6	rl_NumTask	WORD	rl_NumTask	WORD
184	B8	rl_LibList	struct List	rl_LibList	STRUCT
198	C6	rl_NumLib	WORD	rl_NumLib	WORD
200	C8	rl_ClipList	struct List	rl_ClipList	STRUCT
214	D6	rl_NumClip	WORD	rl_NumClip	WORD
216	D8	rl_MsgList	struct List	rl_MsgList	STRUCT
230	E6	rl_NumMsg	WORD	rl_NumMsg	WORD
232	E8	rl_PgmList	struct List	rl_PgmList	STRUCT
246	F6	rl_NumPgm	WORD	rl_NumPgm	WORD
248	F8	rl_TraceCnt	UWORD	rl_TraceCnt	UWORD
250	FA	rl_avail	WORD	rl_avail	WORD

SatisfyMsg

26 bytes

C Include File: devices/clipboard.h
ML Include File: devices/clipboard.i

byte	hex	name	C type	ML type
0	00	sm_Msg	struct Message	STRUCT
20	14	sm_Unit	UWORD	UWORD
22	16	sm_ClipID	LONG	LONG

Screen

346 bytes
(Updated for 3.0)

C Include File: intuition/screens.h
ML Include File: intuition/screens.i

byte	hex	C name	C type	ML name	ML type
0	0000	NextScreen	struct Screen *	sc_NextScreen	APTR
4	0004	FirstWindow	struct Window *	sc_FirstWindow	APTR
8	0008	LeftEdge	WORD	sc_LeftEdge	WORD
10	000A	TopEdge	WORD	sc_TopEdge	WORD
12	000C	Width	WORD	sc_Width	WORD
14	000E	Height	WORD	sc_Height	WORD
16	0010	MouseY	WORD	sc_MouseY	WORD
18	0012	MouseX	WORD	sc_MouseX	WORD
20	0014	Flags	UWORD	sc_Flags	WORD

byte	hex	C name	C type	ML name	ML type
22	0016	Title	UBYTE *	sc_Title	APTR
26	001A	DefaultTitle	UBYTE *	sc_DefaultTitle	APTR
30	001E	BarHeight	BYTE	sc_BarHeight	BYTE
31	001F	BarVBorder	BYTE	sc_BarVBorder	BYTE
32	0020	BarHBorder	BYTE	sc_BarHBorder	BYTE
33	0021	MenuVBorder	BYTE	sc_MenuVBorder	BYTE
34	0022	MenuHBorder	BYTE	sc_MenuHBorder	BYTE
35	0023	WBorTop	BYTE	sc_WBorTop	BYTE
36	0024	WBorLeft	BYTE	sc_WBorLeft	BYTE
37	0025	WBorRight	BYTE	sc_WBorRight	BYTE
38	0026	WBorBottom	BYTE	sc_BorBottom	BYTE
39	0027			sc_KludgeFill00	BYTE
40	0028	Font	struct TextAttr *	sc_Font	APTR
44	002C	ViewPort	struct ViewPort	sc_ViewPort	STRUCT
84	0054	RastPort	struct RastPort	sc_RastPort	STRUCT
184	00B8	BitMap	struct BitMap	sc_BitMap	STRUCT
224	00A2	LayerInfo	struct Layer_Info	sc_layerInfo	STRUCT
326	0146	FirstGadget	struct Gadget *	sc_FirstGadget	APTR
330	014A	DetailPen	UBYTE	sc_DetailPen	BYTE
331	014B	BlockPen	UBYTE	sc_BlockPen	BYTE
332	014C	SaveColor0	UWORD	sc_SaveColor0	WORD
334	014E	BarLayer	struct Layer *	sc_BarLayer	STRUCT
338	0152	ExtData	UBYTE *	sc_ExtData	APTR
342	0156	UserData	UBYTE *	sc_UserData	APTR

Flags

name	bit	decimal	hex
SHOWTITLE	4	16	10
BEEPING	5	32	20
CUSTOMBITMAP	6	64	40
SCREENBEHIND	7	128	80
SCREENQUIET	8	256	100
SCREENHIRES	9	512	200
PENSHARED	10	1024	400
NS_EXTENDED	12	4096	1000
AUTOSCROLL	14	16384	4000

ScreenBuffer

8 bytes
(New to 3.0)

C Include File: intuition/screens.h
ML Include File: intuition/screens.i

byte	hex	name	C type	ML type
0	00	sb_BitMap	struct BitMap *	APTR
4	04	sb_DBufInfo	struct DBufInfo *	APTR

ScreenModePrefs

28 bytes
(New to 3.0)

C Include File: prefs/screenmode.h
ML Include File: prefs/screenmode.i

byte	hex	name	C type	ML type
0	00	smp_Reserved	ULONG[4]	STRUCT
16	10	smp_DisplayID	ULONG	ULONG
20	14	smp_Width	UWORD	UWORD
22	16	smp_Height	UWORD	UWORD
24	18	smp_Depth	UWORD	UWORD
26	1A	smp_Control	UWORD	UWORD

smp_Control

C bit name	C flag name	ML flag name	bit	decimal	hex
SMB_AUTOSCROLL	SMF_AUTOSCROLL	SM,AUTOSCROLL	0	1	01

ScreenModeRequester

46 bytes
(New to 3.0)

C Include File: libraries/asl.h
ML Include File: libraries/asl.i

byte	hex	name	C type	ML type
0	00	sm_DisplayID	ULONG	ULONG
4	04	sm_DisplayWidth	ULONG	ULONG
8	08	sm_DisplayHeight	ULONG	ULONG
12	0C	sm_DisplayDepth	UWORD	UWORD
14	0E	sm_OverscanType	UWORD	UWORD
16	10	sm_AutoScroll	BOOL	UWORD
18	12	sm_BitMapWidth	ULONG	ULONG
20	14	sm_BitMapHeight	ULONG	ULONG
24	18	sm_LeftEdge	WORD	WORD
26	1A	sm_TopEdge	WORD	WORD
28	1C	sm_Width	WORD	WORD

byte	hex	name	C type	ML type
30	1E	sm_Height	WORD	WORD
32	20	sm_InfoOpened	BOOL	BOOL
34	22	sm_InfoLeftEdge	WORD	WORD
36	24	sm_InfoTopEdge	WORD	WORD
38	26	sm_InfoWidth	WORD	WORD
40	28	sm_InfoHeight	WORD	WORD
42	2A	sm_UserData	APTR	APTR

SCSICmd

30 bytes
(New to 2.0)

C Include File: devices/scsidisk.h
ML Include File: devices/scsidisk.i

byte	hex	name	C type	ML type
0	00	scsi_Data	UWORD *	APTR
4	04	scsi_Length	ULONG	ULONG
8	08	scsi_Actual	ULONG	ULONG
12	0C	scsi_Command	UBYTE *	APTR
16	10	scsi_CmdLength	UWORD	UWORD
18	12	scsi_CmdActual	UWORD	UWORD
20	14	scsi_Flags	UBYTE	UBYTE
21	15	scsi_Status	UBYTE	UBYTE
22	16	scsi_SenseData	UBYTE *	APTR
26	1A	scsi_SenseLength	UWORD	UWORD
28	1C	scsi_SenseActual	UWORD	UWORD

Segment

16 bytes
(New to 3.0)

C Include File: dos/dosextens.h
ML Include File: dos/dosextens.i

byte	hex	name	C type	ML type
0	00	seg_Next	BPTR	BPTR
4	04	seg_UC	LONG	LONG
8	08	seg_Seg	BPTR	BPTR
12	0C	seg_Name	UBYTE[4]	STRUCT

Semaphore

(SM in ML)
36 bytes

C Include File: exec/semaphores.h
ML Include File: exec/semaphores.i

byte	hex	C name	C type	ML name	ML type
0	00	sm_MsgPort	struct MsgPort		
34	22	sm_Bids	WORD	SM_BIDS	WORD

SemaphoreMessage

24 bytes
(New to 3.0)

C Include File: exec/semaphores.h
ML Include File: exec/semaphores.i

byte	hex	C name	C type	ML name	ML type
0	00	ssm_Message	struct Message		
20	14	ssm_Semaphore	struct SignalSemaphore *	SSM_SEMAPHORE	APTR

SemaphoreRequest

(SSR in ML)
12 bytes
(Updated for 3.0)

C Include File: exec/semaphores.h
ML Include File: exec/semaphores.i

byte	hex	C name	C type	ML name	ML type
0	00	sr_Link	struct MinNode		
8	08	sr_Waiter	struct Task *	SSR_WAITER	APTR

SerialPrefs

33 bytes
(New to 3.0)

C Include File: prefs/serial.h
ML Include File: prefs/serial.i

byte	hex	name	C type	ML type
0	00	sp_Reserved	LONG[3]	STRUCT
12	0C	sp_Unit0Map	ULONG	ULONG
16	10	sp_BaudRate	ULONG	ULONG
20	14	sp_InputBuffer	ULONG	ULONG
24	18	sp_OutputBuffer	ULONG	ULONG
28	1C	sp_InputHandshake	UBYTE	UBYTE
29	1D	sp_OutputHandshake	UBYTE	UBYTE
30	1E	sp_Parity	UBYTE	UBYTE
31	1F	sp_BitsPerChar	UBYTE	UBYTE
32	20	sp_StopBits	UBYTE	UBYTE

sp_Parity

name	decimal	hex
PARITY_NONE	0	00
PARITY_EVEN	1	01
PARITY_ODD	2	02
PARITY_MARK	3	03
PARITY_SPACE	4	04

sp_InputHandshake
sp_OutputHandshake

name	decimal	hex
HSHAKE_XON	0	00
HSHAKE_RTS	1	01
HSHAKE_NONE	2	02

SGWork

44 bytes
(New to 2.0)

C Include File: intuition/sghooks.h
ML Include File: intuition/sghooks.i

byte	hex	C name	C type	ML name	ML type
0	00	Gadget	struct Gadget *	sgw_Gadget	APTR
4	04	StringInfo	struct StringInfo *	sgw_StringInfo	APTR
8	08	WorkBuffer	UBYTE *	sgw_WorkBuffer	APTR
12	0C	PrevBuffer	UBYTE *	sgw_PrevBuffer	APTR
16	10	Modes	ULONG	sgw_Modes	ULONG
20	14	IEvent	struct InputEvent *	sgw_IEvent	APTR
24	18	Code	UWORD	sgw_Code	UWORD
26	1A	BufferPos	WORD	sgw_BufferPos	WORD

byte	hex	C name	C type	ML name	ML type
28	1C	NumChars	WORD	sgw_NumChars	WORD
30	1E	Actions	ULONG	sgw_Actions	ULONG
34	22	LongInt	LONG	sgw_LongInt	LONG
38	26	GadgetInfo	struct GadgetInfo *	sgw_GadgetInfo	APTR
42	2A	EditOp	UWORD	sgw_EditOp	UWORD

EditOp (sg_EditOp in ML)

name	decimal	hex
EO_NOOP	1	01
EO_DELBACKWARD	2	02
EO_DELFORWARD	3	03
EO_MOVECURSOR	4	04
EO_ENTER	5	05
EO_RESET	6	06
EO_REPLACECHAR	7	07
EO_INSERTCHAR	8	08
EO_BADFORMAT	9	09
EO_BIGCHANGE	10	0A
EO_UNDO	11	0B
EO_CLEAR	12	0C
EO_SPECIAL	13	0D

Modes (sg_Modes in ML)

C flag name	ML flag name	ML bit name	bit	decimal	hex
SGM_REPLACE	SGMF_REPLACE	SGMB_REPLACE	0	1	01
SGM_FIXEDFIELD	SGMF_FIXEDFIELD	SGMB_FIXEDFIELD	1	2	02
SGM_NOFILTER	SGMF_NOFILTER	SGMB_NOFILTER	2	4	04
SGM_EXITHELP	SGMF_EXITHELP	SGMB_EXITHELP	7	128	80

Actions (sg_Actions in ML)

C flag name	ML flag name	ML bit name	bit	decimal	hex
SGA_USE	SGAF_USE	SGAB_USE	0	1	01
SGA_END	SGAF_END	SGAB_END	1	2	02
SGA_BEEP	SGAF_BEEP	SGAB_BEEP	2	4	04
SGA_REUSE	SGAF_REUSE	SGAB_REUSE	3	8	08
SGA_REDISPLAY	SGAF_REDISPLAY	SGAB_REDISPLAY	4	16	10
SGA_NEXTACTIVE	SGAF_NEXTACTIVE	SGAB_NEXTACTIVE	5	32	20
SGA_PREVACTIVE	SGAF_PREVACTIVE	SGAB_PREVACTIVE	6	64	40

SignalSemaphore

(SS in ML)
46 bytes
(Updated for 3.0)

C Include File: exec/semaphores.h
ML Include File: exec/semaphores.i

byte	hex	name	C type	ML name	ML type
0	00	ss_Link	struct Node		STRUCT
14	0E	ss_NestCount	WORD	SS_NESTCOUNT	WORD
16	10	ss_WaitQueue	struct MinList	SS_WAITQUEUE	STRUCT
28	1C	ss_MultipleLink	struct SemaphoreRequest	SS_MULTIPLELINK	STRUCT
40	28	ss_Owner	struct Task *	SS_OWNER	APTR
44	2C	ss_QueueCount	WORD	SS_QUEUECOUNT	WORD

SimpleSprite

12 bytes

C Include File: graphics/sprite.h
ML Include File: graphics/sprite.i

byte	hex	C name	C type	ML name	ML type
0	00	posctldata	UWORD *	ss_posctldata	APTR
4	04	height	UWORD	ss_height	WORD
6	06	x	UWORD	ss_x	WORD
8	08	y	UWORD	ss_y	WORD
10	0A	num	UWORD	ss_num	WORD

SM

see Semaphore

SoftIntList

(SH in ML)
16 bytes

C Include File: exec/interrupts.h
ML Include File: exec/interrupts.i

byte	hex	name	C type	ML name	ML type
0	00	sh_List	struct List		
14	0E	sh_Pad	UWORD	SH_PAD	UWORD

SoundPrefs

284 bytes
(New to 3.0)

C Include File: prefs/sound.h
ML Include File: prefs/sound.i

byte	hex	name	C type	ML type
0	00	sop_Reserved	LONG[4]	STRUCT
16	10	sop_DisplayQueue	BOOL	BOOL
18	12	sop_AudioQueue	BOOL	BOOL
20	14	sop_AudioType	UWORD	UWORD
22	16	sop_AudioVolume	UWORD	UWORD
24	18	sop_AudioPeriod	UWORD	UWORD
26	1A	sop_AudioDuration	UWORD	UWORD
28	1C	sop_AudioFileName	char[256]	STRUCT

sop_AudioType

name	decimal	hex
SPTYPE_BEEP	0	00
SPTYPE_SAMPLE	1	01

SpecialMonitor

58 bytes
(Updated for 3.0)

C Include File: graphics/monitor.h
ML Include File: graphics/monitor.i

byte	hex	C name	C type	ML name	ML type
0	00	spm_Node	struct ExtendedNode		
24	18	spm_Flags	UWORD	spm_Flags	UWORD
26	1A	do_monitor	LONG(*)()	spm_do_monitor	APTR
30	1E	reserved1	LONG(*)()	spm_reserved1	APTR
34	22	reserved2	LONG(*)()	spm_reserved2	APTR
38	26	reserved3	LONG(*)()	spm_reserved3	APTR

byte	hex	C name	C type	ML name	ML type
42	2A	hblank	struct AnalogSignalInterval	spm_hblank	STRUCT
46	2E	vblank	struct AnalogSignalInterval	spm_vblank	STRUCT
50	32	hsync	struct AnalogSignalInterval	spm_hsync	STRUCT
54	36	vsync	struct AnalogSignalInterval	spm_vsync	STRUCT

SpriteDef

8 bytes

C Include File: hardware/custom.h
ML Include File: hardware/custom.i

byte	hex	C name	C type	ML name	ML type
0	00	pos	UWORD	sd_pos	$00
2	02	ctl	UWORD	sd_ctl	$02
4	04	dataa	UWORD	sd_dataa	$04
6	06	datab	UWORD	sd_datab	$06

SrcNode

16 bytes
(New to 2.0)

C Include File: rexx/storage.h
ML Include File: rexx/storage.i

byte	hex	name	C type	ML type
0	00	sn_Succ	struct SrcNode *	APTR
4	04	sn_Pred	struct SrcNode *	APTR
8	08	sn_Ptr	APTR	APTR
12	0C	sn_Size	LONG	LONG

SS

see SignalSemaphore

SSR

see SemaphoreRequest

StackSwapStruct

12 bytes
(New to 3.0)

C Include File: exec/tasks.h
ML Include File: exec/tasks.i

byte	hex	name	type
0	00	stk_Lower	APTR
4	04	stk_Upper	ULONG
8	08	stk_Pointer	APTR

StandardPacket

68 bytes

C Include File: dos/dosextens.h
ML Include File: dos/dosextens.i

byte	hex	name	C type	ML type
0	00	sp_Msg	struct Message	STRUCT
20	14	sp_Pkt	struct DosPacket	STRUCT

Note: This structure was previously found in libraries/dosextens.h (C version) and libraries/dosextens.i (ML version).

StoredProperty

8 bytes
(Updated for 3.0)

C Include File: libraries/iffparse.h
ML Include File: libraries/iffparse.i

byte	hex	C name	C type	ML name	ML type
0	00	sp_Size	LONG	spr_Size	LONG
4	04	sp_Data	APTR	spr_Data	APTR

Note: This structure was previously found in iff/iffparse.h (C version) and iff/iffparse.i (ML version).

StringExtend

36 bytes
(New to 2.0)

C Include File: intuition/sghooks.h
ML Include File: intuition/sghooks.i

byte	hex	C name	C type	ML name	ML type
0	00	Font	struct TextFont *	sex_Font	APTR
4	04	Pens	UBYTE[2]	sex_Pens	STRUCT
6	06	ActivePens	UBYTE[2]	sex_ActivePens	STRUCT
8	08	InitialModes	ULONG	sex_InitialModes	ULONG
12	0C	EditHook	struct Hook *	sex_EditHook	APTR
16	10	WorkBuffer	UBYTE *	sex_WorkBuffer	APTR
20	14	Reserved	ULONG[4]	sex_Reserved	STRUCT

StringInfo

36 bytes
(Updated for 3.0)

C Include File: intuition/intuition.h
ML Include File: intuition/intuition.i

byte	hex	C name	C type	ML name	ML type
0	00	Buffer	UBYTE *	si_Buffer	APTR
4	04	UndoBuffer	UBYTE *	si_UndoBuffer	APTR
8	08	BufferPos	WORD	si_BufferPos	WORD
10	0A	MaxChars	WORD	si_MaxChars	WORD
12	0C	DispPos	WORD	si_DispPos	WORD
14	0E	UndoPos	WORD	si_UndoPos	WORD
16	10	NumChars	WORD	si_NumChars	WORD
18	12	DispCount	WORD	si_DispCount	WORD
20	14	CLeft	WORD	si_CLeft	WORD
22	16	CTop	WORD	si_CTop	WORD
24	18	LayerPtr	struct Layer *	si_LayerPtr	APTR
28	1C	LongInt	LONG	si_LongInt	LONG
32	20	AltKeyMap	struct KeyMap *	si_AltKeyMap	APTR

TabletData

24 bytes
(New to 3.0)

C Include File: intuition/intuition.h
ML Include File: intuition/intuition.i

byte	hex	name	C type	ML type
0	00	td_XFraction	UWORD	UWORD
2	02	td_YFraction	UWORD	UWORD
4	04	td_TabletX	ULONG	ULONG
8	08	td_TabletY	ULONG	ULONG
12	0C	td_RangeX	ULONG	ULONG
16	10	td_RangeY	ULONG	ULONG
20	14	td_TagList	struct TagItem *	APTR

TabletHookData

16 bytes
(New to 3.0)

C Include File: intuition/intuition.h
ML Include File: intuition/intuition.i

byte	hex	name	C type	ML type
0	00	thd_Screen	struct Screen *	APTR
4	04	thd_Width	ULONG	ULONG
8	08	thd_Height	ULONG	ULONG
12	0C	thd_ScreenChanged	LONG	LONG

TAF

see TAvailFonts

TagItem

8 bytes
(New to 2.0)

C Include File: utility/tagitem.h
ML Include File: utility/tagitem.i

byte	hex	name	C type	ML type
0	00	ti_Tag	Tag	ULONG
4	04	ti_Data	ULONG	ULONG

ti_Tag

name	decimal	hex
TAG_DONE	0	00
TAG_END	0	00
TAG_IGNORE	1	01
TAG_MORE	2	02
TAG_SKIP	3	03
TAG_USER	2147783648	80000000

Task

(TC_Struct in ML)
92 bytes
(Updated for 3.0)

C Include File: exec/tasks.h
ML Include File: exec/tasks.i

byte	hex	C name	C type	ML name	ML type
0	00	tc_Node	struct Node	TC_NODE	STRUCTURE
14	0E	tc_Flags	UBYTE	TC_FLAGS	UBYTE
15	0F	tc_State	UBYTE	TC_STATE	UBYTE
16	10	tc_IDNestCnt	BYTE	TC_IDNESTCNT	BYTE
17	11	tc_TDNestCnt	BYTE	TC_TDNESTCNT	BYTE
18	12	tc_SigAlloc	ULONG	TC_SIGALLOC	ULONG
22	16	tc_SigWait	ULONG	TC_SIGWAIT	ULONG
26	1A	tc_SigRecvd	ULONG	TC_SIGRECVD	ULONG
30	1E	tc_SigExcept	ULONG	TC_SIGEXCEPT	ULONG
34	22	tc_TrapAlloc	UWORD	TC_TRAPALLOC	UWORD
36	24	tc_TrapAble	UWORD	TC_TRAPABLE	UWORD
38	26	tc_ExceptData	APTR	TC_EXCEPTDATA	APTR
42	2A	tc_ExceptCode	APTR	TC_EXCEPTCODE	APTR
46	2E	tc_TrapData	APTR	TC_TRAPDATA	APTR
50	32	tc_TrapCode	APTR	TC_TRAPCODE	APTR
54	36	tc_SPReg	APTR	TC_SPREG	APTR
58	3A	tc_SPLower	APTR	TC_SPLOWER	APTR
62	3E	tc_SPUpper	APTR	TC_SPUPPER	APTR
66	42	tc_Switch	VOID(*)()	TC_SWITCH	APTR
70	46	tc_Launch	VOID(*)()	TC_LAUNCH	APTR
74	4A	tc_MemEntry	struct List	TC_MEMENTRY	STRUCT
88	58	tc_UserData	APTR	TC_Userdata	APTR

tc_Flags

C name	ML name	bit	decimal	hex
TF_PROCTIME	T,PROCTIME	0	1	01
TF_ETASK	T,ETASK	3	8	08
TF_STACKCHK	T,STACKCHK	4	16	10
TF_EXCEPT	T,EXCEPT	5	32	20
TF_SWITCH	T,SWITCH	6	64	40
TF_LAUNCH	T,LAUNCH	7	128	80

tc_State

name	bit	decimal	hex
TS_INVALID	0	1	01
TS_ADDED	1	2	02
TS_RUN	2	4	04
TS_READY	3	8	08
TS_WAIT	4	16	10
TS_EXCEPT	5	32	20
TS_REMOVED	6	64	40

TAvailFonts

(TAF in ML)
14 bytes
(New to 2.0)

C Include File: diskfont/diskfont.h
ML Include File: diskfont/diskfont.i

byte	hex	name	C type	ML type
0	00	taf_Type	UWORD	UWORD
2	02	taf_Attr	struct TTextAttr	STRUCT

Note: This structure was previously found in libraries/diskfont.h (C version) and libraries/diskfont.i (ML version).

TC_Struct

see Task

TDU_PublicUnit

(TDC_PUBLICUNIT in ML)
64 bytes
(Updated for 3.0)

C Include File: devices/trackdisk.h
ML Include File: devices/trackdisk.i

byte	hex	C name	C type	ML name	ML type
0	00	tdu_Unit	struct Unit	TDU_UNIT	STRUCTURE
38	26	tdu_Comp01Track	UWORD	TDU_COMP01TRACK	UWORD
40	28	tdu_Comp10Track	UWORD	TDU_COMP10TRACK	UWORD
42	2A	tdu_Comp11Track	UWORD	TDU_COMP11TRACK	UWORD
44	2C	tdu_StepDelay	ULONG	TDU_STEPDELAY	ULONG
48	30	tdu_SettleDelay	ULONG	TDU_SETTLEDELAY	ULONG
52	34	tdu_RetryCnt	UBYTE	TDU_RETRYCNT	UBYTE
53	35	tdu_PubFlags	UBYTE	TDU_PUBFLAGS	UBYTE
54	36	tdu_CurrTrk	UWORD	TDU_CURRTRK	UWORD
56	38	tdu_CalibrateDelay	ULONG	TDU_CALIBRATEDELAY	ULONG
60	3C	tdu_Counter	ULONG	TDU_COUNTER	ULONG

tdu_PubFlags

C flag name	C bit name	ML bit name	bit	decimal	hex
TDPF_NOCLICK	TDPB_NOCLICK	TDP,NOCLICK	0	1	01

TextAttr

8 bytes

C Include File: graphics/text.h
ML Include File: graphics/text.i

byte	hex	name	C type	ML type
0	00	ta_Name	STRPTR	APTR
4	04	ta_YSize	UWORD	UWORD
6	06	ta_Style	UBYTE	UBYTE
7	07	ta_Flags	UBYTE	UBYTE

ta_Style

C name	ML name	bit	decimal	hex
FSF_UNDERLINED	FS,UNDERLINED	0	1	01
FSF_BOLD	FS,BOLD	1	2	02
FSF_ITALIC	FS,ITALIC	2	4	04
FSF_EXTENDED	FS,EXTENDED	3	8	08

ta_Flags

C name	ML name	bit	decimal	hex
FPF_ROMFONT	FP,ROMFONT	0	1	01
FPF_DISKFONT	FP,DISKFONT	1	2	02
FPF_REVPATH	FP,REVPATH	2	4	04
FPF_TALLDOT	FP,TALLDOT	3	8	08
FPF_WIDEDOT	FP,WIDEDOT	4	16	10
FPF_PROPORTIONAL	FP,PROPORTIONAL	5	32	20
FPF_DESIGNED	FP,DESIGNED	6	64	40
FPF_REMOVED	FP,REMOVED	7	128	80

TextExtent

12 bytes
(New to 2.0)

C Include File: graphics/text.h
ML Include File: graphics/text.i

byte	hex	name	C type	ML type
0	00	te_Width	UWORD	UWORD
2	02	te_Height	UWORD	UWORD
4	04	te_Extent	struct Rectangle	STRUCT

TextFont

52 bytes

C Include File: graphics/text.h
ML Include File: graphics/text.i

byte	hex	name	C type	ML type
0	00	tf_Message	struct Message	STRUCTURE
20	14	tf_YSize	UWORD	UWORD
22	16	tf_Style	UBYTE	UBYTE
23	17	tf_Flags	UBYTE	UBYTE
24	18	tf_XSize	UWORD	UWORD
26	1A	tf_Baseline	UWORD	UWORD
28	1C	tf_BoldSmear	UWORD	UWORD
30	1E	tf_Accessors	UWORD	UWORD
32	20	tf_LoChar	UBYTE	UBYTE
33	21	tf_HiChar	UBYTE	UBYTE
34	22	tf_CharData	APTR	APTR

byte	hex	name	C type	ML type
38	26	tf_Modulo	UWORD	WORD
40	28	tf_CharLoc	APTR	APTR
44	2C	tf_CharSpace	APTR	APTR
48	30	tf_CharKern	APTR	APTR

tf_Style

C name	ML name	bit	decimal	hex
FSF_UNDERLINED	FS,UNDERLINED	0	1	01
FSF_BOLD	FS,BOLD	1	2	02
FSF_ITALIC	FS,ITALIC	2	4	04
FSF_EXTENDED	FS,EXTENDED	3	8	08

tf_Flags

C name	ML name	bit	decimal	hex
FPF_ROMFONT	FP,ROMFONT	0	1	01
FPF_DISKFONT	FP,DISKFONT	1	2	02
FPF_REVPATH	FP,REVPATH	2	4	04
FPF_TALLDOT	FP,TALLDOT	3	8	08
FPF_WIDEDOT	FP,WIDEDOT	4	16	10
FPF_PROPORTIONAL	FP,PROPORTIONAL	5	32	20
FPF_DESIGNED	FP,DESIGNED	6	64	40
FPF_REMOVED	FP,REMOVED	7	128	80

TextFontExtension

24 bytes
(New to 2.0)

C Include File: graphics/text.h
ML Include File: graphics/text.i

byte	hex	name	C type	ML type
0	00	tfe_MatchWord	UWORD	UWORD
2	02	tfe_Flags0	UBYTE	UBYTE
3	03	tfe_Flags1	UBYTE	UBYTE
4	04	tfe_BackPtr	struct TextFont *	APTR
8	08	tfe_OrigReplyPort	struct MsgPort *	APTR
12	0C	tfe_Tags	struct TagItem *	APTR
16	10	tfe_OFontPatchS	UWORD *	APTR
20	14	tfe_OFontPatchK	UWORD *	APTR

tfe_Flags0

C flag name	C bit name	ML bit name	bit	decimal	hex
TE0F_NOREMFONT	TE0B_NOREMFONT	TE0,NOREMFONT	0	1	01

TERMARRAY

see IOTArray

TFC

see TFontContents

TFontContents

(TFC in ML)
260 bytes
(New to 2.0)

C Include File: diskfont/diskfont.h
ML Include File: diskfont/diskfont.i

byte	hex	name	C type	ML type
0	0000	tfc_FileName	char [254]	STRUCT
254	00FE	tfc_TagCount	UWORD	UWORD
256	0100	tfc_YSize	UWORD	UWORD
258	0102	tfc_Style	UBYTE	UBYTE
259	0103	tfc_Flags	UBYTE	UBYTE

Note: This structure was previously found in libraries/diskfont.h (C version) and libraries/diskfont.i (ML version).

timerequest

(TIMEREQUEST in ML)
40 bytes

C Include File: devices/timer.h
ML Include File: devices/timer.i

byte	hex	C name	C type	ML name	ML type
0	00	tr_node	struct IORequest		STRUCTURE
32	20	tr_time	struct timeval	IOTV_TIME	STRUCT

timeval

(TIMEVAL in ML)
8 bytes

C Include File: devices/timer.h
ML Include File: devices/timer.i

byte	hex	name	C type	ML name	ML type
0	00	tv_secs	ULONG	TV_SECS	ULONG
4	04	tv_micro	ULONG	TV_MICRO	ULONG

Tool

8 bytes
(New to 3.0)

C Include File: datatypes/datatypes.h
ML Include File: datatypes/datatypes.i

byte	hex	name	C type	ML type
0	00	tn_Which	UWORD	UWORD
2	02	tn_Flags	UWORD	UWORD
4	04	tn_Program	STRPTR	APTR

tn_Which

name	decimal	hex
TW_INFO	1	01
TW_BROWSE	2	02
TW_EDIT	3	03
TW_PRINT	4	04
TW_MAIL	5	05

tn_Flags

name	decimal	hex
TF_SHELL	1	01
TF_WORKBENCH	2	02
TF_RX	3	03
TF_LAUNCH_MASK	15	0F

ToolNode

26 bytes
(New to 3.0)

C Include File: datatypes/datatypes.h
ML Include File: datatypes/datatypes.i

byte	hex	name	C type	ML type
0	00	tn_Node	struct Node	STRUCT
14	0E	tn_Tool	struct Tool	STRUCT
22	16	tn_Length	ULONG	ULONG

TmpRas

8 bytes

C Include File: graphics/rastport.h
ML Include File: graphics/rastport.i

byte	hex	C name	C type	ML name	ML type
0	00	RasPtr	BYTE *	tr_RasPtr	APTR
4	04	Size	LONG	tr_Size	LONG

TP_AmigaXIP

8 bytes
(New to 3.0)

C Include File: resources/card.h
ML Include File: resources/card.i

byte	hex	C name	C type	ML name	ML type
0	00	TPL_CODE	UBYTE		
0	00			tp_AmigaXIP	STRUCT
1	01	TPL_LINK	UBYTE		
2	02	TP_XIPLOC	UBYTE[4]	STRUCT	
6	06	TP_XIPFLAGS	UBYTE	STRUCT	
7	07	TP_XIPRESRV	UBYTE	STRUCT	

TTextAttr

12 bytes
(New to 2.0)

C Include File: graphics/text.h
ML Include File: graphics/text.i

byte	hex	name	C type	ML type
0	00	tta_Name	STRPTR	APTR
4	04	tta_YSize	UWORD	UWORD
6	06	tta_Style	UBYTE	UBYTE
7	07	tta_Flags	UBYTE	UBYTE
8	08	tta_Tags	struct TagItem *	APTR

UCopList

12 bytes

C Include File: graphics/copper.h
ML Include File: graphics/copper.i

byte	hex	C name	C type	ML name	ML type
0	00	Next	struct UCopList *	ucl_Next	APTR
4	04	FirstCopList	struct CopList *	ucl_FirstCopList	APTR
8	08	CopList	struct CopList *	ucl_CopList	APTR

Unit

(UNIT in ML)
38 bytes

C Include File: exec/devices.h
ML Include File: exec/devices.i

byte	hex	C name	C type	ML name	ML type
0	00	unit_MsgPort	struct MsgPort	UNIT_MSGPORT	STRUCT
34	20	unit_flags	UBYTE	UNIT_FLAGS	UBYTE
35	21	unit_pad	UBYTE	UNIT_pad	UBYTE
36	22	unit_OpenCnt	UWORD	UNIT_OPENCNT	UWORD

unit_flags

C name	ML name	bit	decimal	hex
UNITF_ACTIVE	UNIT,ACTIVE	0	1	01
UNITF_INTASK	UNIT,INTASK	1	2	02

UtilityBase

36 bytes
(New to 3.0)

C Include File: utility/utility.h
ML Include File: utility/utility.i

byte	hex	name	C type	ML type
0	00	ub_LibNode	struct Library	STRUCTURE
34	22	ub_Language	UBYTE	UBYTE
35	23	ub_Reserved	UBYTE	UBYTE

View

18 bytes
(Updated for 3.0)

C Include File: graphics/view.h
ML Include File: graphics/view.i

byte	hex	C name	C type	ML name	ML type
0	00	ViewPort	struct ViewPort *	v_ViewPort	LONG
4	04	LOFCprList	struct cprlist *	v_LOFCprList	LONG
8	08	SHFCprList	struct cprlist *	v_SHFCprList	LONG
12	0C	DyOffset	WORD	v_DyOffset	WORD
14	0E	DxOffset	WORD	v_DxOffset	WORD
16	10	Modes	UWORD	v_Modes	WORD

Modes

C name	ML name	bit	decimal	hex
GENLOCK_VIDEO	GENLOCK_VIDEO	1	2	0002
LACE	V_LACE	2	4	0004
PFBA	V_PFBA	6	64	0040
EXTRA_HALFBRITE		7	128	0080
GENLOCK_AUDIO		8	256	0100
DUALPF	V_DUALPF	10	1024	0400

C name	ML name	bit	decimal	hex
HAM	V_HAM	11	2048	0800
VP_HIDE		13	8192	2000
SPRITES	V_SPRITES	14	16384	4000
HIRES	V_HIRES	15	32768	8000

ViewExtra

34 bytes
(Updated for 3.0)

C Include File: graphics/view.h
ML Include File: graphics/view.i

byte	hex	C name	C type	ML name	ML type
0	00	n	struct ExtendedNode		
24	18	View	View *	ve_View	APTR
28	1C	Monitor	struct MonitorSpec *	ve_Monitor	APTR
32	20	TopLine	UWORD	ve_TopLine	WORD

ViewPort

40 bytes
(Updated for 3.0)

C Include File: graphics/view.h
ML Include File: graphics/view.i

byte	hex	C name	C type	ML name	ML type
0	00	Next	struct ViewPort *	vp_Next	LONG
4	04	ColorMap	struct ColorMap *	vp_ColorMap	LONG
8	08	DspIns	struct CopList *	vp_DspIns	LONG
12	0C	SprIns	struct CopList *	vp_SprIns	LONG
16	10	ClrIns	struct CopList *	vp_ClrIns	LONG
20	14	UCopIns	struct UCopList *	vp_UCopIns	LONG
24	18	DWidth	WORD	vp_DWidth	WORD
26	1A	DHeight	WORD	vp_DHeight	WORD
28	1C	DxOffset	WORD	vp_DxOffset	WORD
30	1E	DyOffset	WORD	vp_DyOffset	WORD
32	20	Modes	UWORD	vp_Modes	WORD
34	22	SpritePriorities	UBYTE	vp_SpritePriorities	BYTE
35	23	reserved	UBYTE	vp_reserved	BYTE
36	24	RasInfo	struct RasInfo *	vp_RasInfo	APTR

Modes

C name	ML name	bit	decimal	hex
GENLOCK_VIDEO	GENLOCK_VIDEO	1	2	0002
LACE	V_LACE	2	4	0004
PFBA	V_PFBA	6	64	0040
EXTRA_HALFBRITE		7	128	0080
GENLOCK_AUDIO		8	256	0100
DUALPF	V_DUALPF	10	1024	0400
HAM	V_HAM	11	2048	0800
VP_HIDE		13	8192	2000
SPRITES	V_SPRITES	14	16384	4000
HIRES	V_HIRES	15	32768	8000

ViewPortExtra

54 bytes
(Updated for 3.0)

C Include File: graphics/view.h
ML Include File: graphics/view.i

byte	hex	C name	C type	ML name	ML type
0	00	n	struct ExtendedNode		
24	18	ViewPort	struct ViewPort *	vpe_ViewPort	APTR
28	1C	DisplayClip	struct Rectangle	vpe_DisplayClip	STRUCT
36	24	VecTable	APTR	vpe_VecTable	APTR
40	28	DriverData	APTR[2]	vpe_DriverData	STRUCT
48	30	Flags	UWORD	vpe_Flags	UWORD
50	32	Origin	Point[2]	vpw_Origin	STRUCT

VoiceHeader

20 bytes
(New to 3.0)

C Include File: datatypes/soundclass.h
ML Include File: datatypes/soundclass.i

byte	hex	name	type
0	00	vh_OneShotHiSamples	ULONG
4	04	vh_RepeatHiSamples	ULONG
8	08	vh_SamplesPerHiCycle	ULONG
12	0C	vh_SamplesPerSec	UWORD

byte	hex	name	type
14	0E	vh_Octaves	UBYTE
15	0F	vh_Compression	UBYTE
16	10	vh_Volume	ULONG

vh_Compression

name	decimal	hex
CMP_NONE	0	00
CMP_FIBDELTA	1	01

VS

see VSprite

VSprite

(VS in ML)
60 bytes

C Include File: graphics/gels.h
ML Include File: graphics/gels.i

byte	hex	C name	C type	ML name	ML type
0	00	NextVSprite	struct VSprite *	vs_NextVSprite	APTR
4	04	PrevVSprite	struct VSprite *	vs_PrevVSprite	APTR
8	08	DrawPath	struct VSprite *	vs_DrawPath	APTR
12	0C	ClearPath	struct VSprite *	vs_ClearPath	APTR
16	10	OldY	WORD	vs_Oldy	WORD
18	12	OldX	WORD	vs_Oldx	WORD
20	14	Flags	WORD	vs_Flags	WORD
22	16	Y	WORD	vs_Y	WORD
24	18	X	WORD	vs_X	WORD
26	1A	Height	WORD	vs_Height	WORD
28	1C	Width	WORD	vs_Width	WORD
30	1E	Depth	WORD	vs_Depth	WORD
32	20	MeMask	WORD	vs_MeMask	WORD
34	22	HitMask	WORD	vs_HitMask	WORD
36	24	ImageData	WORD *	vs_ImageData	APTR
40	26	BorderLine	WORD *	vs_BorderLine	APTR
44	2A	CollMask	WORD *	vs_CollMask	APTR
48	30	SprColors	WORD *	vs_SprColors	APTR
52	34	VSBob	struct Bob *	vs_VSBob	APTR
56	38	PlanePick	BYTE	vs_PlanePick	BYTE
57	39	PlaneOnOff	BYTE	vs_PlaneOnOff	BYTE
58	3A	VUserExt	VUserStuff		

Flags

C name	ML name	bit	decimal	hex
VSPRITE	VS,VSPRITE	0	1	0001
SAVEBACK	VS,SAVEBACK	1	2	0002
OVERLAY	VS,OVERLAY	2	4	0004
MUSTDRAW	VS,MUSTDRAW	3	8	0008
BACKSAVED	VS,BACKSAVED	8	256	0100
BOBUPDATE	VS,BOBUPDATE	9	512	0200
GELGONE	VS,GELGONE	10	1024	0400
VSOVERFLOW	VS,VSOVERFLOW	11	2048	0800

WBArg

8 bytes

C Include File: workbench/startup.h
ML Include File: workbench/startup.i

byte	hex	name	C type	ML type
0	00	wa_Lock	BPTR	BPTR
4	04	wa_Name	BYTE *	APTR

WBPatternPrefs

24 bytes
(New to 3.0)

C Include File: prefs/wbpattern.h
ML Include File: prefs/wbpattern.i

byte	hex	name	C type	ML type
0	00	wbp_Reserved	ULONG[4]	STRUCT
16	10	wbp_Which	UWORD	UWORD
18	12	wbp_Flags	UWORD	UWORD
20	14	wbp_Revision	BYTE	BYTE
21	15	wbp_Depth	BYTE	BYTE
22	16	wbp_DataLength	UWORD	UWORD

wbp_Which

name	decimal	hex
WBP_ROOT	0	00
WBP_DRAWER	1	01
WBP_SCREEN	2	02

wbp_Flags

C flag name	ML flag name	bit	decimal	hex
WBPF_PATTERN	WBP,PATTERN	0	1	01
WBPF_NOREMAP	WBP,NOREMAP	4	16	10

WBStartup

40 bytes

C Include File: workbench/startup.h
ML Include File: workbench/startup.i

byte	hex	name	C type	ML type
0	00	sm_Message	struct Message	STRUCT
20	14	sm_Process	struct MsgPort *	APTR
24	18	sm_Segment	BPTR	BPTR
28	1C	sm_NumArgs	LONG	LONG
32	20	sm_ToolWindow	char *	APTR
36	24	sm_ArgList	struct WBArg *	APTR

Window

136 bytes
(Updated for 3.0)

C Include File: intuition/intuition.h
ML Include File: intuition/intuition.i

byte	hex	C name	C type	ML name	ML type
0	00	NextWindow	struct Window *	wd_NextWindow	APTR
4	04	LeftEdge	WORD	wd_LeftEdge	WORD
6	06	TopEdge	WORD	wd_TopEdge	WORD
8	08	Width	WORD	wd_Width	WORD
10	0A	Height	WORD	wd_Height	WORD
12	0C	MouseY	WORD	wd_MouseY	WORD
14	0E	MouseX	WORD	wd_MouseX	WORD
16	10	MinWidth	WORD	wd_MinWidth	WORD
18	12	MinHeight	WORD	wd_MinHeight	WORD
20	14	MaxWidth	UWORD	wd_MaxWidth	WORD
22	16	MaxHeight	UWORD	wd_MaxHeight	WORD
24	18	Flags	ULONG	wd_Flags	LONG
28	1C	MenuStrip	struct Menu *	wd_MenuStrip	APTR
32	20	Title	UBYTE *	wd_Title	APTR
36	24	FirstRequest	struct Requester *	wd_FirstRequest	APTR

byte	hex	C name	C type	ML name	ML type
40	28	DMRequest	struct Requester *	wd_DMRequest	APTR
44	2C	ReqCount	WORD	wd_ReqCount	WORD
46	2E	WScreen	struct Screen *	wd_WScreen	APTR
50	32	RPort	struct rastPort *	wd_RPort	APTR
54	36	BorderLeft	BYTE	wd_BorderLeft	BYTE
55	37	BorderTop	BYTE	wd_BorderTop	BYTE
56	38	BorderRight	BYTE	wd_BorderRight	BYTE
57	39	BorderBottom	BYTE	wd_BorderBottom	BYTE
58	3A	BorderRPort	struct RastPort *	wd_BorderRPort	APTR
62	3E	FirstGadget	struct Gadget *	wd_FirstGadget	APTR
66	42	Parent	struct Window *	wd_Parent	APTR
70	46	Descendant	struct Window *	wd_Descendant	APTR
74	4A	Pointer	UWORD*	wd_Pointer	APTR
78	4E	PtrHeight	BYTE	wd_PtrHeight	BYTE
79	4F	PtrWidth	BYTE	wd_PtrWidth	BYTE
80	50	XOffset	BYTE	wd_XOffset	BYTE
81	51	YOffset	BYTE	wd_YOffset	BYTE
82	52	IDCMPFlags	ULONG	wd_IDCMPFlags	ULONG
86	56	UserPort	struct MsgPort *	wd_UserPort	APTR
90	5A	WindowPort	struct MsgPort *	wd_WindowPort	APTR
94	5E	MessageKey	struct IntuiMessage *	wd_MessageKey	APTR
98	62	DetailPen	UBYTE	wd_DetailPen	BYTE
99	63	BlockPen	UBYTE	wd_BlockPen	BYTE
100	64	CheckMark	struct Image *	wd_CheckMark	APTR
104	68	ScreenTitle	UBYTE *	wd_ScreenTitle	APTR
108	6C	GZZMouseX	WORD	wd_GZZMouseX	WORD
110	6E	GZZMouseY	WORD	wd_GZZMouseY	WORD
112	70	GZZWidth	WORD	wd_GZZWidth	WORD
114	72	GZZHeight	WORD	wd_GZZHeight	WORD
116	74	ExtData	UBYTE *	wd_ExtData	APTR
120	78	UserData	BYTE *	wd_UserData	APTR
124	7C	WLayer	struct Layer *	wd_WLayer	APTR
128	80	IFont	struct TextFont *	wd_IFont	APTR
132	84	MoreFlags	ULONG	wd_MoreFlags	ULONG

Flags

name	bit	decimal	hex
WINDOWSIZING	0	1	00000001
WINDOWDRAG	1	2	00000002
WINDOWDEPTH	2	4	00000004
WINDOWCLOSE	3	8	00000008
SIZEBRIGHT	4	16	00000010
SIZEBBOTTOM	5	32	00000020
BACKDROP	8	256	00000100

name	bit	decimal	hex
REPORTMOUSE	9	512	00000200
GIMMEZEROZERO	10	1024	00000400
BORDERLESS	11	2048	00000800
ACTIVATE	12	4096	00001000
WINDOWACTIVE	13	8192	00002000
INREQUEST	14	16384	00004000
MENUSTATE	15	32768	00008000
RMBTRAP	16	65536	00010000
NOCAREREFRESH	17	131072	00020000
WINDOWREFRESH	24	16777216	01000000
WBENCHWINDOW	25	33554432	02000000
WINDOWTICKED	26	67108864	04000000

IDCMPFlags

name	bit	decimal	hex
SIZEVERIFY	0	1	00000001
NEWSIZE	1	2	00000002
REFRESHWINDOW	2	4	00000004
MOUSEBUTTONS	3	8	00000008
MOUSEMOVE	4	16	00000010
GADGETDOWN	5	32	00000020
GADGETUP	6	64	00000040
REQSET	7	128	00000080
MENUPICK	8	256	00000100
CLOSEWINDOW	9	512	00000200
RAWKEY	10	1024	00000400
REQVERIFY	11	2048	00000800
REQCLEAR	12	4096	00001000
MENUVERIFY	13	8192	00002000
NEWPREFS	14	16384	00004000
DISKINSERTED	15	32768	00008000
DISKREMOVED	16	65536	00010000
WBENCHMESSAGE	17	131072	00020000
ACTIVEWINDOW	18	262144	00040000
INACTIVEWINDOW	19	524288	00080000
DELTAMOVE	20	1048576	00100000
VANILLAKEY	21	2097152	00200000
INTUITICKS	22	4194304	00400000
LONELYMESSAGE	31	2147483648	80000000

XLN

see ExtendedNode

XRef

32 bytes
(New to 3.0)

C Include File: libraries/amigaguide.h
ML Include File: libraries/amigaguide.i

byte	hex	name	C type	ML type
0	00	xr_Node	struct Node	STRUCT
14	0E	xr_Pad	UWORD	UWORD
16	10	xr_DF	struct DocFile *	APTR
20	14	xr_File	STRPTR	APTR
24	18	xr_Name	STRPTR	APTR
28	1C	xr_Line	LONG	LONG

3
Hardware Registers

Function calls and system structures allow you to control your Amiga, but it's the computer's hardware that really makes things happen. This chapter explains how the hardware works, how the Amiga's operating systems uses the hardware, and how you can bypass the function calls and system structures to program the hardware directly.

Note that this chapter only discusses the operation of the Amiga's OCS (Old Chip Set) and ECS (Enhanced Chip Set) because all future Amigas are guaranteed to support these hardware configurations. **You should never directly access the AGA (Advanced Graphics Architecture) hardware found on newer model Amigas.** If you do, your program will almost certainly fail to work on future Amigas. Commodore is committed to updating the Amiga with more powerful graphics and sound capabilities. The only way Commodore can do this while ensuring downward software compatibility is if you, the programmer, use the operating system to access the new features provided by the AGA chip set. However, if you must program down to the "bare metal," the information provided here tells you how to do so in a way that is compatible with all model Amigas—past, present, and future.

What's In a Name?

Contributing to the Amiga's uniqueness are the human names given to many of the Amiga's integrated circuits, otherwise known as chips. Foremost among the Amiga's hardware components are the custom chips Agnus, Denise, and Paula. They're called *custom* because they were designed exclusively for the Amiga by Amiga engineers. No other computer uses these chips.

Agnus is the most famous of the Amiga's custom chips. It contains the computer's *blitter* (the part of the computer that generates almost all graphics and animation), *copper* (a video coprocessor responsible for keeping the Amiga's screen display up to date), and *DMA* circuitry (which gives the custom chips direct access to the computer's chip RAM).

Denise performs many of the Amiga's low-level video functions, taking raw bitplane and sprite data and translating it into the RGB signals your monitor understands. Denise also helps out with reading the computer's game ports.

Paula is the Amiga's main I/O chip. It handles disk operations, serial transmis-

sions, some game-port I/O, and, most notably, the digitizing and playback of sound. Paula is also in charge of *interrupts*—events that cause the computer to stop what it's currently doing so it can perform such important duties as polling the mouse, reading the keyboard, and switching the attention of the 68000 to the next multitasking program in need of processor time.

Besides the custom chips, the Amiga uses two 8250 CIA (Complex Interface Adapter) chips. These chips help with serial, parallel, keyboard, and game-port I/O, as well as operating the Amiga's floppy disk controller. While the Amiga contains several other electronic components, only the custom chips and CIA chips can be directly accessed by the programmer.

The Amiga's hardware is accessed via *registers*—locations within a chip that control the chip's operation or return status information about that chip. A register is accessed by reading or writing to its address, just as if you were reading or writing to a location in RAM.

Bits, Bytes, and Words

Before you can understand how to manipulate the Amiga's hardware registers, you must first understand the binary numbering system and the relationship between bits, bytes, and words.

Bits are the computer's smallest unit of information. They can contain only the number 0 or 1, but when combined they can form any number needed. For example, if two bits are combined, the number increases: 00, 01, 10, 11. That makes four possible combinations. And if a third bit is added—000, 001, 010, 011, 100, 101, 110, 111—you get eight different combinations.

When eight bits are put together, the number of combinations increases to 256. These eight bits are called a *byte* and can be used to represent the numbers from 0 to 255. When 16 bits (two bytes) are put together, you get a *word*. A word has 65,536 different combinations and can be used to represent the numbers from 0 to 65,535.

This system of numbering is called *binary* (or base two). It works much like the decimal (base ten) system. In the base ten numbering system, the right-most digit is known as the one's place, and holds a number from 0 to 9. The next digit to the left is known as the ten's place, which also holds a number from 0 to 9 and represents the number of times the one's place has counted past 9 (the number of tens).

In the binary system, there is a one's place, then a two's place, a four's place, and so on. The bits are counted from right to left, starting with bit 0. Here are the values of each bit in a word:

Bit	Value	Bit	Value
0	1	8	256
1	2	9	512
2	4	10	1024
3	8	11	2048
4	16	12	4096
5	32	13	8192
6	64	14	16384
7	128	15	32768

If all the bits are added together (1 + 2 + 4 + 8 + 16 + 32 + 64 + 128 + 256 . . .), they total 65,535, which is the maximum value of one word. What if you need to count higher than 65,535? Use two words. When two words (32 bits) are grouped together, they form a *long word* and can be used to hold a number from 0 to 4,294,967,295.

A *byte* is the smallest addressable memory location on the Amiga, and the Amiga can access up to nine megabytes of RAM (9,437,184 different memory locations). Many hardware registers are used as *pointers* (a number that specifies the address of a memory location, such as the address of sprite data or sound data). Because most hardware registers are a word in length and a word holds a maximum value of only 65,535, two consecutive registers are often combined to form one *long-word pointer*. The first register holds the address's *high word* (bits 16 through 31) and the second register holds the address's *low word* (bits 0 through 15). Together they can point to any addressable location in the Amiga.

Hexadecimal

One other numbering system used in speaking about computers is the *hexadecimal* (base 16) system. Each hexadecimal digit can count a number from 0 to 15. Since the highest numeric digit is 9, the alphabet must be used: A = 10, B = 11, C = 12, D = 13, E = 14, and F = 15. With just two digits, 256 combinations are possible (16 times 16). That means one byte can be represented by just two hexadecimal digits, each of which stands for four bits. It takes four hexadecimal digits to represent a word and eight hexadecimal digits to represent a long word.

Hexadecimal (*hex* for short) has become the standard numbering system when referring to memory locations on the Amiga. Because of this, all register addresses found in this book are given in hex. Other values may be given in decimal or even binary. A dollar sign ($) has been placed in front of all hex numbers. Note that in C hex numbers begin with a 0x. In machine language hex numbers begin with a dollar sign ($), just like the hex numbers found within the text of this book.

Accessing the Hardware

The Amiga's hardware is accessed via locations $BFD000$BFDF00 for CIA chip B, $BFE001–$BFEF01 for CIA chip A, and $DFF000-$DFF1BE for the custom chips. Reading or writing to these addresses affects the corresponding hardware register. In the case of the CIA chips, each register is a byte long and may be read from or written to using a single address.

The custom chip registers are different, however. They are all a word in length and each one has a separate address for reading and a separate address for writing. In fact, some of these registers have only one address, meaning you can only read from or write to them. Most of the custom chip registers belong to just one chip, but in certain cases a register may belong to two, or even all three, custom chips.

> **Warning:** You should never attempt to read a write-only register. Doing so will place a random value in that register, possibly causing your computer to lock up. For the same reason, you should never use the machine language instructions BCLR and BSET on a write-only register, since these two instructions perform a read operation prior to clearing or setting the specified bit.

The end of this chapter contains a hardware *memory map*—a list of registers arranged in order of their address location. (To look up a register by name, see the Hardware Index that follows this chapter.) At the beginning of each entry, you'll find the register's hex address, its name, a brief description of its function, its read/write status, and to which chip or chips it belongs. For example, the header for the DMACON register looks like this:

$DFF096 DMACON

DMA Control

Status: Write-Only. Chip: Agnus/Denise/Paula

Below this header is a complete description of the register as well as a discussion of each of the register's bits, if appropriate.

Many registers work together, such as the blitter registers. In this case, you'll find the heading *Location Range* followed by the address range of the related registers and a description of what the registers do. The registers are grouped under one heading like this when it would be meaningless to explain one register without explaining them all. For detailed information, each register still has its own separate entry found below the *Location Range* heading.

Accessing a hardware register is easy in machine language. The programming

examples in this chapter first equate the register's name to its address and then read or write to that address directly, just as if it were a location in RAM. For example, here's how you might use the DMACON and COLOR00 registers to blank the screen from machine language:

```
DMACON equ $DFF096 ;Define DMA Control register
COLOR00 equ $DFF180 ;Define Color register 0
  move.w #$0000,COLOR00 ;Make background color black
  move.w #$0180,DMACON ;Disable copper and bitplane DMA
```

When copper and bitplane DMA is turned off, the Amiga cannot display graphics so the screen goes blank.

In C, you just declare a pointer to the desired register address and use the pointer to access the register:

```
UWORD *DMACON = (UWORD *)0xDFF096; /* Declare DMA Control register */
UWORD *COLOR00 = (UWORD *)0xDFF180; /* Declare Color register 0 */
  *COLOR00=0; /* Make background color black */
  *DMACON=0x0180 /* Disable copper and bitplane DMA */
```

Most registers should be declared as a pointer to a word since most registers are a word in length. CIA registers should be declared as pointers to a byte. Registers that contain pointers themselves should be declared as pointers to pointers, as in the case of the AUD0LC register which holds the pointer to the first byte of channel 0's waveform data. Here's how you might declare this register:

```
BYTE **AUD0LC = (BYTE **)0xDFF0A0;
```

Programmers can optionally use hardware include files, to predefine some of the labels given to registers. In this book, however, the registers are defined only as needed, and they're defined in the manner shown above.

Boolean Logic: AND, OR, and EOR

Each bit within a hardware register may control a separate function on the Amiga. You will often see references to setting bit 6 equal to one, or setting bit 3 equal to zero. This can be done by adding or subtracting the bit value for that particular bit from the value of the whole register.

Adding or subtracting the bit value works only if you know the status of that bit already. If bit 4 is off, and you add 16 (the value of bit 4) to the register, it will turn bit

4 on. But if it was on already, adding 16 would turn bit 4 off and another bit on. This is where the boolean logic functions—AND, OR, and EOR—come in handy.

AND is usually used to zero out (or mask) unwanted bits. When you AND a number with another, a 1 will appear in the resulting number only if identical bits in the ANDed numbers had been set to 1. For example, if you wanted to turn off bit 4 in a byte-long register that contains 154, you could AND it with 239 (which is the maximum bit combination minus the bit value to be masked, 255-16):

```
      10011010 = 154 ($9A)
AND   11101111 = 239 ($EF)
=     10001010 = 138 ($8A)
```

By using the AND function, nothing would be harmed if we tried to turn off a bit that wasn't on. You can always turn off a bit by using the formula VALUE AND (255-BITVALUE) for byte-long registers or VALUE AND (65535-BITVALUE) for word-long registers. Remember, there must be a 1 in the same bit of both numbers in order for the same bit in the result to be equal to 1.

Turning a bit on is performed by the OR function. OR puts a 1 in the bit of the resulting number if there's a 1 in the same bit in either of the two numbers. For example, to turn bit 4 back on in the number 138, we would use the statement 138 OR 16 (the bit value for the bit we want to turn on):

```
      10001010 = 138 ($8A)
OR    00010000 = 16  ($10)
=     10011010 = 154 ($9A)
```

Again, no harm would be done if the bit was already on. A bit can always be turned on with the formula VALUE OR BITVALUE.

The third operation, EOR, is used to reverse the value of a bit. If the bit of the second number holds a 1, it will reverse the value of the corresponding bit in the first number. Therefore, to switch all of the bits, you can EOR a number with 255:

```
      10011010 = 154 ($9A)
EOR   11111111 = 255 ($FF)
=     01100101 = 101 ($65)
```

Notice that this produces the complement of the original number (255 minus the number). Anytime you wish to flip a bit from 0 to 1 or from 1 to 0, you can EOR it with the value of that bit.

So far we've been talking about the boolean operators AND, OR, and EOR using generic names. In the C programming language, these operators are represented by symbols, not words. In C, the AND, OR, and EOR operators are represented by the ampersand (&), vertical bar (|), and caret (^) symbol, respectively. So to clear bit 4 in the number 154, you would use the instruction **RESULT = 154 & 239**.

AND, OR, and EOR are the actual names of the instructions in machine language. So to toggle all the bits in the number 154, you might use the machine language commands **MOVE #154,d0** and **EOR #255,d0**, which would leave the result in the byte portion of register d0. (Machine language also has the specialized instructions BCLR and BSET for clearing and setting bits. As noted previously, however, these instructions should not be used on the Amiga's write-only registers.)

It's important to understand these concepts; in one way or another, everything in the computer is controlled by boolean logic.

The SET/CLR Bit

While looking through the custom chip registers you'll find several that contain a bit labelled SET/CLR. This is a special control bit that allows you to set and clear the register's other bits without using the AND and OR functions. SET/CLR is always bit 15 (the highest bit in the register).

What happens to a register that contains a SET/CLR bit depends on the way you set this bit. If it is set to 0, any other bit that is written to with a 1 will be cleared (equal to 0). If bit 15 is set to 1, any other bit that is written to with a 1 will be set (equal to 1). All bits written to with a 0 remain unaffected. So to clear bit 0, you'd write a $0001 to the register. Writing the value $7FFF would clear all of the bits. To set bit 0 equal to 1, you'd write a $8001 to the register. Writing the value $FFFF would set them all.

The SET/CLR bit is useful only on write-only registers. On read-only registers, this bit always returns a value of 0.

Warning

The information given in the following register map is provided so you can have a better understanding of how the Amiga works. Programming the hardware directly is *not* recommended. It goes against most of the rules set by Commodore. Worst of all, a program that writes directly to the hardware will not multitask correctly since it may disrupt other programs trying to access the same registers. The Amiga's hardware should be controlled by using the function calls provided by the Amiga's operating system.

In many cases, you can't even control a hardware device without disabling the operating system first. The disk controller is one such device. If you try to program disk drives directly without disabling the operating system, you'll simply end up fighting the

operating system for control, and possibly destroying valuable disk data at the same time.

If, on the other hand, your program does not need to multitask, has no need for the Amiga's operating system, and requires as much speed and control as possible, programming the hardware directly may be exactly what you want. It provides the fastest, most efficient method of getting things done. But remember, it also makes your programs unfriendly to, and incompatible with, other Amiga programs.

Custom Chips

$DFF000 BLTDDAT

Blitter Destination Data (early read dummy address)
Status: DMA-Only. Chip: Agnus
Just before a word of data is moved into RAM by the blitter, it's stored in this register. This register cannot be accessed by the microprocessor or by the copper. It's used exclusively by blitter DMA. Writing to this address does nothing. Reading this address returns a somewhat random number.

$DFF002 DMACONR

DMA Enable Read
Status: Read-Only. Chip: Agnus/Paula

Bit 0: AUD0EN: 1 = Audio channel 0 DMA enabled
Bit 1: AUD1EN: 1 = Audio channel 1 DMA enabled
Bit 2: AUD2EN: 1 = Audio channel 2 DMA enabled
Bit 3: AUD3EN: 1 = Audio channel 3 DMA enabled
Bit 4: DSKEN: 1 = Disk DMA enabled
Bit 5: SPREN: 1 = Sprite DMA enabled
Bit 6: BLTEN: 1 = Blitter DMA enabled
Bit 7: COPEN: 1 = Copper DMA enabled
Bit 8: BPLEN: 1 = Bitplane DMA enabled
Bit 9: DMAEN: Master DMA off switch
(0 = DMA for all channels is disabled)
Bit 10: BLTPRI: Blitter priority
(1 = Blitter has full priority over the CPU,
0 = Blitter has partial priority over the CPU);
Bit 11: Unused

Bit 12: Unused
Bit 13: BZERO: 1 = If the last blitter operation output zeros only (no bits were set in the destination RAM)
Bit 14: BBUSY: 1 = Blitter is currently performing an operation (it's "blitting")
Bit 15: SET/CLR: Not used by DMACONR

This register reflects the on/off status of the Amiga's DMA.

Bit 0--0. See register DMACON ($DFF096).

Bit 13. This bit equals 1 if the last blitter operation results in all zeros. This feature can be useful for detecting collisions between two images. Simply set up the blitter so the destination is the combination of images A and B (minterm $C0), turn the blitter on, and then check this bit. If this bit is clear (equal to 0), the images touch. When using the blitter for this purpose, it's best to clear the USED bit (bit 8) in the BLTCON0 register ($DFF040). This prevents the blitter from outputting any data, thus speeding up the operation and saving you the trouble of allocating a destination buffer in chip RAM.

Bit 14 This bit equals 1 if the blitter is currently performing an operation. This bit equals 0 when the blitter is free for use, at which time the INTREQ register's blitter-ready bit (bit 6) is set to 1. You should always wait for the blitter to finish its current task before you give it something else to do. The following machine language program uses this bit to do just that:

```
DMACONR equ $DFF002
BlitterWait:
 btst #6,DMACONR
Wait:
 btst #6,DMACONR
 bne  Wait
 rts
```

This subroutine waits until the blitter is free and then returns. Because of timing problems with Amigas that use the faster 68020, 68030, and 68040 processors, you must wait a short while before testing the BBUSY bit for a nonzero value. The first BTST instruction in the subroutine above provides this delay. If you use the blitter functions provided by the graphics.library, you don't need to use such a subroutine, since the graphics.library functions do this for you.

$DFF004 VPOSR

Vertical Beam Position Read
Status: Read-Only. Chip: Agnus

Bit 0:	V8: The high bit of the vertical beam position (1 = Vertical beam position is greater than 255)
Bits 1-14:	Unused
Bit 15:	LOF: Interlace long-frame flag (1 = A long frame is being drawn; 0 = A short frame is being drawn)

When you use this register in conjunction with the next one, VHPOSR, you can read the current vertical position of the video beam. The vertical beam position is stored as a 9-bit number. The low 8 bits are found in VHPOSR. Bit 0 of this register contains the high bit.

In interlace mode, bit 15 of this register tells you whether the video hardware is drawing a long frame or a short frame. This bit holds a 1 when a long frame is being drawn and a 0 when a short frame is being drawn. (In noninterlace modes, this bit remains equal to 1.)

To understand the relationship between video beam positions, long frames, short frames, and interlace mode, you must first understand how the Amiga's video hardware generates a picture.

Your television or monitor uses an electron gun (RGB monitors use three guns—one for red, one for green, and one for blue) to paint a beam of pixels across the screen, starting in the top left corner and moving in a straight line to the right. When this beam has been drawn, the electron gun turns off and returns to the left side of the screen (this event is called a horizontal blank). It then moves down, turns back on, and draws another line. Every beam on the display is drawn from left to right. On a standard, noninterlace Amiga display, there are 262 such lines, 200 of which form the visible screen area. To keep the image on screen, this 262-line display is redrawn 60 times a second. (The video display on European PAL systems contain 312 lines, which are redrawn 50 times a second.) One such scan of the screen is called a *frame*.

Interlace mode doubles the vertical resolution of your Amiga's display. Because most monitors are incapable of drawing more than 350 video beams in 1/60 of a second, the Amiga uses a simple trick to achieve the higher resolution: Each video frame contains only half of the screen's lines. In the first frame, only the even-numbered beams are drawn—beams 0, 2, 4, 6 . . . 524. In the next frame, only the odd-numbered beams are drawn—beams 1, 3, 5, 7. . . 523. (Interlaced PAL displays contain 625 beams, numbered 0-624). These frames continue to alternate, thus "interlacing" the even- and odd-numbered video beams. Interlace screens tend to flicker, however,

because each beam is on the screen for only half the usual amount of time.

A frame that draws even-numbered beams is called a *long frame*. Conversely, a *short frame* is a frame that draws odd-numbered beams. Long frames earn their name from the fact that they contain one more video beam than short frames. Because of this, it takes a slightly longer amount of time to draw a long frame.

$DFF006 VHPOSR

Vertical/Horizontal Beam Position Read
Status: Read-Only. Chip: Agnus

Bits 0-7: H1-H8: Video beam's horizontal position specified in increments of two low-resolution pixels
Bits 8-15: V0-V7: Low 8 bits of video beams vertical position

Bits 0–7. The low 8 bits in this register specify the horizontal position of the video beam. This horizontal value is accurate to only two low-resolution pixels (four high-resolution pixels).

The horizontal value returned by VHPOSR can vary between 0 and 227 ($0 and $E3). On a normal display, the first visible position is 64. The last visible position on a standard screen is 224. The horizontal value increases as the video beam moves right across the screen. When it reaches 227—which is when the video beam is about 10 low-resolution pixels left of the right-hand border—it resets to 0 and resumes incrementing. As a result, horizontal location 0 is on the far right side of the screen, near the edge of the border. Positions 225-227 and 0-6 make up the right border; positions 7-45 occur during the horizontal blank; and positions 46-63 make up the left border.

A funny thing happens on the way from horizontal position 227 to horizontal position 0: The vertical beam position reflected by bits 8-16 increases by 1, even though horizontal positions 227 and 0 are found on the same physical screen line. So, when specifying the screen position in *(x,y)* format, location (0,2) directly follows location (227,1). It's not until location (46,2) that the video beam actually moves down a line. This is important to remember when you're writing a copper list that must wait for an exact screen location, because the copper uses the VPOSR and VHPOSR registers to determine the current position of the video beam.

Bits 8–15. These bits reflect the low 8 bits of the video beam's vertical position. The high bit is found in register VPOSR ($DFF004). Since these two registers are found in consecutive memory locations, it's easy to read them as one long word. To read the video beam's vertical position in machine language, for example, use the following instructions:

```
VPOSR equ $DFF004

  move.l VPOSR,d0       ;Read VPOSR and VHPOSR into d0 as one long word
  and.l #$0001FF00,d0   ;Get vertical position only
  lsr.l #8,d0           ;Shift d0 right 8 times to right-justify value
```

After executing this code, register d0 contains the vertical position of the video beam. This results in a number ranging between 0 and 262 (0 and 312 for PAL systems). In interlace mode, short frames never return a value higher than 261 (311 for PAL).

The screen's normal viewing area is contained in vertical positions 44-243 (this changes, of course, whenever someone lowers the screen using the drag bar). Anything above or below this range is either part of the border or entirely off the screen. Positions 244-262 make up the bottom border, positions 0-21 occur during the vertical blank, and positions 22-43 make up the top border.

The vertical value returned by this register always reflects the number of noninterlace scan lines. To calculate the physical vertical beam position for an interlace screen, take the value returned by registers VPOSR and VHPOS, multiply it by 2, and, if VPOSR's LOF bit is clear, add 1. The following machine language subroutine will do this for you, returning the result in the word portion of register d0 (the register's lower 16 bits):

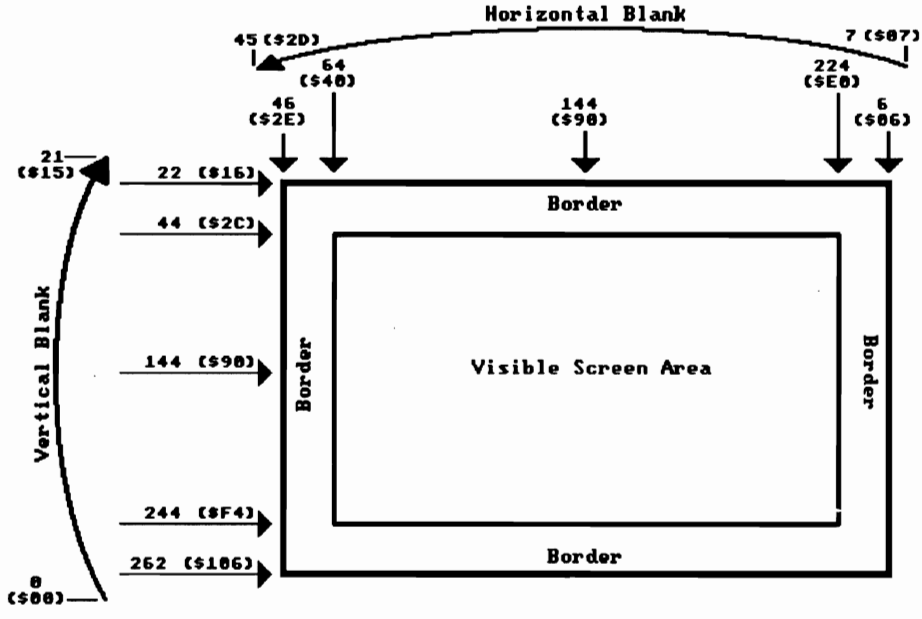
```
VPOSR equ $DFF004

  move.l  VPOSR,d0      ;Read VPOSR and VHPOSR into d0 as one long word
  and.l  #$8001FF00,d0  ;Get vertical position and LOF bit
  lsr.l  #7,d0          ;Right-justify and multiply by 2
  btst   #24,d0         ;Test LOF bit (recently shifted into bit 24)
  bne.s  Done           ;Exit if long frame (beams 0, 2, 4...)
  addq.w #1,d0          ;Add 1 if short frame (beams 1, 3, 5...)
Done:
  rts                   ;Return from subroutine
```

Important: If bit 3 in register BPLCON0 ($DFF100) is set, the Amiga's light pen hardware is activated and both VPOSR and VHPOSR return the pixel position of the light pen—not the current position of the video beam. The position of the light pen is updated during each vertical blank.

Figure 3-1 shows how both the horizontal and vertical positions returned by VPOSR and VHPOSR relate to a standard NTSC video display.

Figure 3-1. VPOSR and VHPOSR Screen Positions



Note that the location and size of the visible screen area changes according to the values stored in the DIWSTRT (Display Window Start) and DIWSTOP (Display Window Stop) registers ($DFF08E and $DFF090). These registers are changed by Preferences each time you adjust the display-centering gadet. Keep this in mind when referring to Figure 3-1.

$DFF008 DSKDATR

Disk Data Read (early read dummy address)

Status: DMA-Only. Chip: Paula

This register holds the two bytes of data currently being read from or written to disk by the Amiga's DMA. This register cannot be accessed by the microprocessor or by the copper. It's used exclusively by disk DMA. Writing to this address does nothing. Reading this address returns a somewhat random number.

Location Range: $DFF00A-$DFF00C

Joystick/Mouse Input Ports

These registers handle input for both mice and joysticks. They also return the status of the fire buttons on paddles. Register JOY0DAT handles port 1 and register JOY1DAT handles port 2.

Mouse Input. When you move your Amiga's mouse, it sends a series of electronic pulses to the computer. These pulses are converted into a number that increments and decrements in relation to the distance moved. Bits 0-7 hold the mouse's X (horizontal) movement counter and Bits 8-15 hold the mouse's Y (vertical) movement counter.

The X counter decreases as you move the mouse left and increases as you move the mouse right. The Y counter decreases as you move the mouse up and increases as you move it down. By reading a counter, waiting a little while, and then reading it again, you can see how far the mouse moved simply by calculating the difference between the two values. Both counters change at a rate of approximately 200 counts per inch.

Since each counter is only 8 bits long, their values vary between 0 and 255. When a counter goes above 255 (overflows), it wraps around to 0 and continues to count up. Similarly, when a counter goes below 0 (underflows), it wraps around to 255 and continues to count down.

If you're not careful, you could easily translate a counter overflow as a drastic movement downward or to the right; or you could confuse a counter underflow as a drastic movement upward or to the left. To avoid this, you must check the counter values at frequent, regular intervals.

It's recommended you check the counters at least 60 times per second, say during each vertical blank. It's a safe assumption that the average human will not move the mouse more than 3 feet per second, which is how fast you'd have to move it to get a counter change of more than 127 in only 1/60 of a second. So if the difference in counter values is greater than +127 or less than -127, you know that an underflow or an overflow has occurred.

For example, let's say that the current Y counter value is 13 and the last Y counter value was 200. Because the difference between these two values is -187, an overflow must have taken place (or the user moved the mouse up almost an entire inch in just 1/60 of a second). In this case, the real difference in counter values is a downward movement of 69, less than 1/2 of an inch. The basic logic here is:

Distance = CurrentValue - LastValue
if Distance < -127 then Distance = 256 + Distance
if Distance > 127 then Distance = Distance -256

where Distance returns the counter's true change in value.

The contents of the JOYxDAT registers may be set using the JOYTEST register ($DFF036).

The status of the left mouse button is returned by bit 6 in the CIA-A PRA register ($BFE001). The right mouse button is handled by bit 10 in the POTGOR register ($DFF016). For mice plugged into port 2, the status of the left mouse button is returned by bit 7 in the CIA-A PRA register and the right mouse button is handled POTGOR's bit 14.

Joystick Input. Joysticks consist of five switches, one for each direction and one for the fire button. Each direction switch is given a separate bit in the JOYxDAT registers. If the joystick is moved left or right, that switch is pressed and the corresponding bit is set to 1. Bit 1 returns the status of the right switch and bit 9 returns the status of the left switch.

Reading the up and down joystick position is slightly more complex. These switches are read via two bits that you must exclusive OR with each other to get the proper reading (EOR in machine language, XOR in BASIC, and ^ in C). If bit 0 EOR bit 1 returns a 1, the joystick is being pushed up. If bit 8 EOR bit 9 returns a 1, the joystick is being pushed down.

The following C program uses the JOY1DAT register to print the current status of the joystick plugged into port 2:

```
short joy, *JOYDAT = (short *)0xDFF00C;

void main()
{
  joy = *JOYDAT;                        /* read register contents */
  if (joy & 2)                          /* check bit 1 */
     printf("right\n");
  if (joy & 512)                        /* check bit 9 */
     printf("left\n");
  if ((joy & 1) ^ (joy & 2)>>1)         /* check bit 0 EOR bit 1 */
     printf("down\n");
  if ((joy & 256) ^ (joy & 512)>>1) /* check bit 8 EOR bit 9 */
     printf("up\n");
}
```

For port 1, the joystick's fire button status is returned by 6 in the CIA-A PRA register ($BFE001). For port 2, bit 7 in register CIA-A PRA returns the joystick's fire button status. Pressing a joystick's fire button is the same as pressing the left button on a mouse.

$DFF00A JOY0DAT
Joystick/Mouse Port 1 Data
Status: Read-Only. Chip: Denise
For Mouse Use:
Bits 0-7: Horizontal position counter
Bits 8-15: Vertical position counter

For Joystick Use:
Bit 0 EOR Bit 1: Down (1 = Joystick pushed down)
Bit 1: Right (1 = Joystick pushed right)
Bit 8 EOR Bit 9: Up (1 = Joystick pushed up)
Bit 9: Left (1 = Joystick pushed left)

For Paddle Use:
Bit 1: Right paddle's fire button (1 = Fire button pressed)
Bit 9: Left paddle's fire button (1 = Fire button pressed)

Joystick/Mouse read register for port 1.

$DFF00C JOY1DAT
Joystick/Mouse Port 2 Data
Status: Read-Only. Chip: Denise

For Mouse Use:
Bits 0-7: Horizontal position counter
Bits 8-15: Vertical position counter

For Joystick Use:

Bit 0 EOR Bit 1: Down (1 = Joystick pushed down)
Bit 1: Right (1 = Joystick pushed right)
Bit 8 EOR Bit 9: Up (1 = Joystick pushed up)
Bit 9: Left (1 = Joystick pushed left)

For Paddle Use:
Bit 1: Right paddle's fire button (1 = Fire button pressed)
Bit 9: Left paddle's fire button (1 = Fire button pressed)

Joystick/Mouse read register for port 2.

$DFF00E CLXDAT

Collision Data

Status: Read-Only. Chip: Denise

Bit 0: 1 = Even bitplane collided with odd bitplane
Bit 1: 1 = Odd bitplane collided with sprite 0 (or 1)
Bit 2: 1 = Odd bitplane collided with sprite 2 (or 3)
Bit 3: 1 = Odd bitplane collided with sprite 4 (or 5)
Bit 4: 1 = Odd bitplane collided with sprite 6 (or 7)
Bit 5: 1 = Even bitplane collided with sprite 0 (or 1)
Bit 6: 1 = Even bitplane collided with sprite 2 (or 3)
Bit 7: 1 = Even bitplane collided with sprite 4 (or 5)
Bit 8: 1 = Even bitplane collided with sprite 6 (or 7)
Bit 9: 1 = Sprite 0 (or sprite 1) collided with sprite 2 (or sprite 3)
Bit 10: 1 = Sprite 0 (or sprite 1) collided with sprite 4 (or sprite 5)
Bit 11: 1 = Sprite 0 (or sprite 1) collided with sprite 6 (or sprite 7)
Bit 12: 1 = Sprite 2 (or sprite 3) collided with sprite 4 (or sprite 5)
Bit 13: 1 = Sprite 2 (or sprite 3) collided with sprite 6 (or sprite 7)
Bit 14: 1 = Sprite 4 (or sprite 5) collided with sprite 6 (or sprite 7)
Bit 15: Unused

The Amiga's hardware automatically detects collisions between graphic objects. You just tell it what type of collisions you're interested in, and it does the rest. You can detect collision between sprites, collisions between sprites and bitplane graphics, and even collisions between odd- and even-numbered bitplanes (useful for playfield animation). This register tells you when and what type of a collision has taken place. When you read this register, all of the bits are reset to 0, so be sure to save its contents if you wish to check for more than one type of collision.

A sprite collision occurs when one of its non-background pixels contacts something, such as another sprite. Collision detection is always enabled for even numbered sprites, but not for odd numbered sprites. You can enable collision detection for odd numbered sprites using the CLXCON register. However, every two sprites share the same collision-detection bit. So if collision detection is enabled for sprite 1 and bit 9 of CLXDAT signals a collision, there's no way to tell whether sprite 0 or sprite 1 was the cause.

Bitplane collisions are extremely flexible. You can disable collisions with any of the six possible bitplanes, and you can specify that collisions only take place with pixels of a certain color. See register CLXCON ($DFF098) for more information.

$DFF010 ADKCONR

Audio/Disk Control Read
Status: Read-Only. Chip: Paula

Bit 0:	ATVOL0: 1 = Audio channel 0 modulates the volume of channel 1
Bit 1:	ATVOL1: 1 = Audio channel 1 modulates the volume of channel 2
Bit 2:	ATVOL2: 1 = Audio channel 2 modulates the volume of channel 3
Bit 3:	ATVOL3: 1 = Output of audio channel 3 disabled
Bit 4:	ATPER0: 1 = Audio channel 0 modulates the period of channel 1
Bit 5:	ATPER1: 1 = Audio channel 1 modulates the period of channel 2
Bit 6:	ATPER2: 1 = Audio channel 2 modulates the period of channel 3
Bit 7:	ATPER3: 1 = Output of audio channel 3 disabled
Bit 8:	FAST: 1 = MFM, two-microseconds-per-bit disk operation; 0 = GCR, four-microseconds-per-bit disk operation (not used by AmigaDOS)
Bit 9:	MSBSYNC: 1 = GCR-format synchronization for disk operations enabled (not used by AmigaDOS)
Bit 10:	WORDSYNC: 1 = Disk controller synchronizing on the data word found in the DSKSYNC register ($DFF07E)
Bit 11:	UARTBRK: 1 = RS-232 break-signal occuring on the serial port's TXD line
Bit 12:	MFMPREC: 1 = MFM disk format selected; 0 = GCR disk format selected (not used by AmigaDOS)
Bits 13-14:	PRECOMP: Disk precompensation time (00 = None; 01 = 140 nanoseconds; 10 = 280 nanoseconds; 11 = 560 nanoseconds)
Bit 15:	SET/CLR: Not used by ADKCONR

This register returns the modulation status of the audio channels, the current working mode of the disk controller, and the break status of the Amiga's RS-232 serial port. It is the read version of the ADKCON register ($DFF09E). See that register for a complete explanation of each bit.

Location Range: $DFF012-$DFF014

Paddle/Proportional Joystick Input Ports

These registers allow you to read proportional input devices such as the paddles and joysticks sold for Apple II and MS-DOS computers. These input devices use variable resistors (also known as *potentiometers* or *pots*) with electrical resistance that changes

when you turn the paddle's knob or move the proportional joystick in any direction. The Amiga measures this resistance as it updates the screen display and returns a numeric value corresponding to this resistance in the POTxDAT registers. The lower the value, the less electrical resistance was encountered.

These registers work in tandem with the POTGO register ($DFF034). During the vertical blanking period, you store a 1 into POTGO, wait till the next vertical blank, and then read one of these registers. The register you read depends upon which port the joystick or paddle is plugged into. What results is a number between 0 and 255 appearing in each of the register's two bytes. The rotation position of the right paddle is returned in the registers' high byte; the rotation of the left paddle is returned in the registers' low byte. When using joysticks, the high byte returns the vertical position and the low byte returns the horizontal position.

With paddles, a 0 tells you that the knob is turned all the way counterclockwise, and a 255 tells you that the knob is turned all the way clockwise. With joysticks, a 0 tells you that the joystick is being pushed all the way down or to the left, and a 255 tells you that the joystick is being pushed all the way up or to the right.

Not all paddles and proportional joysticks return values as high as 255, however. The highest value obtainable depends on the rating of the potentiometer used by the paddle or joystick. Paddles designed for the Commodore 64, for example, generally return values that range between 0 and 64. Most proportional joysticks designed for MS-DOS computers return a maximum value of only 18 or so. For the Amiga, Commodore recommends using paddles and joysticks that use a 470K Ohm potentiometer with a 10-percent tolerance rating.

$DFF012 POTODAT

Pot (Paddle/Proportional Joystick) Port 1 Data Read
Status: Read-Only. Chip: Paula

For Paddle Use:

Bits 0-7: Left paddle counter (0 = Turned completely counterclockwise)
Bits 8-15: Right paddle counter (0 = Turned completely counterclockwise)

For Proportional Joystick Use:

Bits 0-7: Horizontal counter (0 = Joystick being pushed left)
Bits 8-15: Vertical counter (0 = Joystick being pushed down)

$DFF014 POT1DAT

Pot (Paddle/Proportional Joystick) Port 2 Data Read
Status: Read-Only. Chip: Paula

For Paddle Use:
Bits 0-7: Left paddle counter (0 = Turned completely clockwise)
Bits 8-15: Right paddle counter (0 = Turned completely clockwise)
For Proportional Joystick Use:
Bits 0-7: Horizontal counter (0 = Joystick being pushed left)
Bits 8-15: Vertical counter (0 = Joystick being pushed down)

$DFF016 POTGOR

Pot Port Data Read
Status: Read-Only. Chip: Paula

Bit 0: START: Trigger bit for starting pot counters. Always reads 0
Bits 1-7: Unused (reserved for chip identification)
Bit 8: DATLX: Current state of pin 5 on game port 1
(1 = Positive voltage; 0 = Zero volts)
Bit 9: OUTLX: Output enable for pin 5 on game port 1. Always reads 0
Bit 10: DATLY: Current state of pin 9 on game port 1
(1 = Positive voltage; 0 = Zero volts)
Bit 11: OUTLY: Output enable for pin 9 on game port 1. Always reads 0
Bit 12: DTARX: Current state of pin 5 on game port 2
(1 = Positive voltage; 0 = Zero volts)
Bit 13: OUTRX: Output enable for pin 5 on game port 2. Always reads 0
Bit 14: DATRY: Current state of pin 9 on game port 2
(1 = Positive voltage; 0 = Zero volts)
Bit 15: OUTRY: Output enable for pin 9 on game port 2. Always reads 0

This is the read address for the POTGO register ($DFF034). Most of the bits in this register are write-only and always return a 0 when read. Only bits 8, 10, 12, and 14 return valid data. The remaining bits are controlled via the POTGO register. See location $DFF034 for more information on these bits.

The Amiga's game ports are quite versatile. Pins 5 and 9 of either port can be programmed to receive and send digital data. This register allows you to read the current high-low state of these pins.

Simply read the desired pin's DATxx bit to determine its status: 1 means the pin carries positive voltage, usually +5 volts or more; 0 means the pin carries no or negative

voltage. The positive voltage sent to these pins should never exceed 5 volts or 400 milliamps.

The right mouse button is wired to game port pins 8 and 9. When the button is pressed, these two pins connect. Pin 8 is connected to ground, so any voltage that pin 9 might carry is shorted to ground whenever the right mouse button is pressed.

To sense whether the right mouse button is being pressed on a mouse plugged into game port 1, you must first set pin 9 to output (write a 1 to POTGO's OUTLY bit) and send voltage to this pin (write a 1 to POTGO's DATLY bit). This is done by storing a $0C00 into POTGO. (For mice plugged into game port 2, use bits OUTRY and DATRY instead.) Finally, wait about 300 microseconds and then read the status of pin 9 via the DATLY bit in POTGOR. If this bit equals 1, the pin must not be grounded and, therefore, the mouse button must not be pressed. If this bit equals 0, the pin must be grounded and, therefore, the button must be pressed.

Because Intuition automatically handles game port 1 mouse events, you'll have to turn off multitasking if you wish to control this port directly via hardware. Even accessing game port 2 can cause some confusion, since any manipulation of the POTGO register affects all of the register's bits. Such confusion usually results in the Amiga thinking that the right mouse button is being clicked when it really isn't.

The following machine language subroutine waits until the right mouse button in pressed on a mouse plugged into game port 2 (the port that the mouse is normally not plugged into):

```
VPOSR  equ $DFF004
POTGOR equ $DFF016
POTGO  equ $DFF034

WaitRMB
   bsr     WaitVertB      ;wait for vertical blank
   move.w  #$C000,POTGO   ;send 5 volts out pin 9
   bsr     WaitVertB      ;wait for another vertical blank
   btst    #6,POTGOR      ;read status of pin 9
   bne     WaitRMB        ;loop if it carries voltage
   rts

WaitVertB
   move.l  VPOSR,d0       ;read VPOSR and VHPOSR registers
   and.l   #$0001FF00,d0  ;mask out vertical beam position
   bne     WaitVertB      ;wait until vertical beam position is 0
   rts
```

$DFF018 SERDATR

Serial Data Input and Status Read
Status: Read-Only. Chip: Paula

Bits 0-7: DB0-DB7: Byte of data read from serial port
Bit 8: DB8 or STP: Ninth data bit or stop bit
Bit 9: STP: Stop bit if serial port is set up for nine data bits
Bit 10: Unused
Bit 11: RXD: Current status of serial port's RXD line (pin 3)
Bit 12: TSRE: 1 = Transmit shift register is empty
Bit 13: TBE: 1 = Transmit buffer is empty
Bit 14: RBF: 1 = Receive buffer is full
Bit 15: OVRUN: 1 = Receive buffer has overrun

There are three registers that control the Amiga's serial port: SERDAT ($DFF030), which is where you store data to be output; SERPER ($DFF032), which allows you to set the serial port's baud rate and number of data bits; and this register, SERDATR, which acts as a status register and receives all incoming data.

Data sent through the serial port is output one bit at a time—low-bit first, high-bit last. To announce the arrival of new data, a start bit equal to 1 is sent out. Then, depending on the protocol used, eight or nine data bits are sent. A stop bit—which, like the start bit, is equal to 1—is sent to flag the end of data.

To transfer data, the Amiga uses special shift registers. The shift register that sends data shifts right, pushing the lowest bit out the serial port's TXD line until the register is empty. Incoming data is received via the RXD line, where it is shifted right into another shift register until the stop bit is encountered. You cannot access the shift registers directly. Instead, you use backup registers.

SERDAT is the backup register for outgoing data (see location $DFF030 for more information). SERDATR, the register explained here, is the backup register for incoming data. Once the shift register receives all eight or nine data bits and the stop bit, its contents are moved into bits 0-9 of this register and the RBF (Receive Buffer Full) bit is set equal to 1. (Note that the start bit is thrown away.) If you neglect to read SERDATR before the shift register fills up again, an overrun error occurs.

Here's a bit-by-bit explanation of the SERDATR register:

Bits 0-7. These bits comprise the serial port's receive buffer; they hold the most recent byte of data received. When new data arrives, the RBF bit (bit 14) is set equal to 1. After reading these bits, you should clear the RBF bit in the INTREQ register ($DFF09C) to avoid a false receive-buffer overrun error.

Bit 8. When receiving nine data-bit transmissions, this bit contains the ninth bit

of data. When receiving eight data-bit transmissions, this bit holds the stop bit, in which case it is always equal to 1. Bit 15 of the SERPER register ($DFF032) determines whether the serial port is set up for eight or nine data bits.

Bit 9. This holds the stop bit when receiving serial transmissions containing nine data bits.

Bit 11. This bit allows you to read the serial port's RXD line directly. RXD is pin 3 on the serial port; it receives all incoming data, including the start and stop bits. When pin 3 is high (carries positive voltage), this bit holds a 0. Negative voltage signifies a 1, so this bit holds a 1 when pin 3 has no or negative voltage. With transfer rates as high as 31.25K baud, the RXD line can change quite frequently.

Bit 12. If both the transmit shift-register and the SERDAT register is empty (that is, any data written to SERDAT has already been output by the shift register), this bit equals 1.

Bit 13. Unlike the bit above, this bit is set equal to 1 as soon as the data in the SERDAT register moves into the shift register. It remains equal to 1 until SERDAT is written to again. You can use this bit to detect when SERDAT is ready to receive more data. This event can also trigger a level 1 interrupt. See the INTENR register ($DFF01C) for more information.

Bit 14. This bit is a mirror of bit 11 in the INTREQR register ($DFF01E). As soon as SERDATR's bits 0-9 receive new data, this bit is set equal to 1. After reading bits 0-9, you should clear the RBF bit in the INTREQ register ($DFF09C) to avoid a false receive-buffer overrun error (see the following bit description).

Bit 15. If new data arrives in bits 0-9 before it is read, an overrun error occurs and this bit is set equal to 1.

$DFF01A DSKBYTR

Disk Data Byte and Disk Status Read
Status: Read-Only. Chip: Paula

Bits 0-7:	DATA: Byte of data read from disk
Bit 8-11:	Unused
Bit 12:	WORDEQUAL: 1 = Disk sync pattern found
Bit 13:	DISKWRITE: 1 = Disk controller is writing data; 0 = Disk controller is reading data
Bit 14:	DMAON: 1 = Disk DMA is active
Bit 15:	BYTEREADY: 1 = Bits 0-7 of this register hold new data

Both this register and the PRA register ($BFE001) in the CIA-A chip provide status information on the Amiga's floppy disk controller. This register also holds the byte

currently being read from disk. The Amiga uses this register to synchronize the computer's microprocessor with the rotation of the disk.

Bits 0-7. The disk controller reads data from disk one byte at a time, placing the data here as it goes along. Every time a new byte is loaded into this register, the BYTEREADY bit (bit 15) is set equal to 1. Considering the fact that disk data can be transferred to and from memory automatically through DMA (see locations $DFF020-$DFF026), reading data from disk using these eight bits is neither necessary or recommended.

Bit 12. This bit is set when the disk controller finds a sync pattern—a 16-bit word that signifies the beginning of data. AmigaDOS uses the number $4489 as its sync pattern. You specify the synchronization value by writing it to the DSKSYNC register ($DFF07E) after setting the WORDSYNC bit (bit 10) in the ADKCON register ($DFF09E). Using the INTREQ and INTENA registers, you can generate a level 5 interrupt whenever the synchronization maker is found. Note that when the sync pattern is found, WORDEQUAL holds a 1 for only 2 microseconds.

Bit 13. This bit returns the current status of the DSKLEN register's WRITE bit, telling you whether the disk controller is in read or write mode. A 0 means the controller is in read mode; a 1 means the controller is in write mode.

Bit 14. If disk DMA is transferring data, this bit equals 1. In order for this to be true, the DMAENA bit in both the DSKLEN register ($DFF024) and the DMACON register ($DFF096) must be set.

Bit 15. When a new byte of data makes its way into bits 0-7, this bit is set equal to 1. This bit is automatically cleared whenever you read this register.

$DFF01C INTENAR

Interrupt Enable Read
Status: Read-Only. Chip: Paula

Bit 0: TBE: 1 = Serial transmit buffer empty interrupt enabled (level 1)
Bit 1: DSKBLK: 1 = Disk block done interrupt enabled (level 1)
Bit 2: SOFT: 1 = Software interrupts enabled (level 1)
Bit 3: PORTS: 1 = CIA-A and expansion port interrupts enabled (level 2)
Bit 4: COPER: 1 = Copper interrupt enabled (level 3)
Bit 5: VERTB: 1 = Vertical blank interrupt enabled (level 3)
Bit 6: BLIT: 1 = Blitter ready interrupt enabled (level 3)
Bit 7: AUD0: 1 = Audio channel 0 interrupt enabled (level 4)
Bit 8: AUD1: 1 = Audio channel 1 interrupt enabled (level 4)
Bit 9: AUD2: 1 = Audio channel 2 interrupt enabled (level 4)

Bit 10: AUD3: 1 = Audio channel 3 interrupt enabled (level 4)
Bit 11: RBF: 1 = Serial receive buffer full interrupt enabled (level 5)
Bit 12: DSKSYN: 1 = Disk sync pattern found interrupt enabled (level 5)
Bit 13: EXTER: 1 = CIA-B and expansion port interrupts enabled (level 6)
Bit 14: INTEN: Master interrupt off switch (0 = All interrupts listed above disabled)
Bit 15: SET/CLR: Not used by INTENAR

The Amiga's hardware is capable of generating 14 maskable interrupts. An interrupt is a signal given to the microprocessor that tells it to stop executing the current program and to work on another program, known as an *interrupt processing routine,* for a short period of time. After finishing the interrupt routine, the computer goes back to executing the main program just as if it had never left.

Computer interrupts are often compared to telephone calls that divert us from our current task—programming, reading a book, watching TV, and so on—allowing us to continue our task only after the telephone call is over. The interrupts controlled by this register are called *maskable* because we can disable them so they won't bother us. Disabling an interrupt is similar to unplugging the telephone.

This register tells you which interrupts are enabled (allowed to take place). To enable or disable an interrupt, use this register's write address, INTENA ($DFF09A). To force an interrupt to take place, use register INTREQ ($DFF09C). To find out which type of interrupt is or is not taking place, use register INTREQR ($DFF01E).

The Amiga's 680x0 microprocessor supports seven different interrupts, where each one is assigned a unique level number. The higher the level number, the higher the interrupt's priority. If two interrupts occur at the same time, the interrupt with the highest priority is handled first. A level 7 interrupt has such a high priority that it cannot be disabled. This type of interrupt is called *non-maskable*. Therefore, the 680x0 supports only six maskable interrupts, levels 1-6.

In order for the Amiga to manage 14 maskable interrupts when its microprocessor recognizes only six, some interrupts were assigned the same 680x0 level number. (The level numbers assigned to each interrupt are shown in parentheses in the bit table above.) All audio interrupts, for example, initiate a level 4 680x0 interrupt.

To prioritize interrupts that share the same level number, Exec has assigned each interrupt a pseudo priority number. Again, the higher this number, the higher the interrupt's priority. The following table lists the level number and pseudo priority number for each of the 14 maskable interrupts:

Interrupt	680x0 Level	Exec's Pseudo Priority
software	1	1
disk block done	1	2
transmit buffer empty	1	3
CIA-A or expansion bus pin 19	2	1
copper	3	1
vertical blank	3	2
blitter done	3	3
audio channel 2	4	1
audio channel 0	4	2
audio channel 3	4	3
audio channel 1	4	4
read buffer full	5	1
disk sync pattern found	5	2
CIA-B or expansion bus pin 22	6	1

If you've thrown out the operating system and plan to handle interrupts completely on your own, note that interrupt processing routines are always executed in 680x0 supervisor mode. Also, any registers that may be altered by your interrupt routine should be saved—usually by pushing them onto the stack—at the start of the routine and then restored—usually by pulling them off of the stack—at the end of the routine. At a minimum, your interrupt routine should check the INTREQR register to see what caused the interrupt and then clear the corresponding bit in the INTREQ register to clear that interrupt. Use the 680x0 RTE instruction to return from an interrupt.

For a highly-detailed explanation of Amiga interrupts, see Carl Sassenrath's *Guru's Guide To The Commodore Amiga: Meditation #1—Interrupts* (Sassenrath Research, P.O. Box 1510, Ukiah, CA 95482).

Bits 0-13 of this register allow you to check if any of the 14 maskable interrupts are enabled. If an interrupt's corresponding bit is set to 1, that interrupt is enabled. Organized by their corresponding bit number, the conditions that could cause a maskable interrupt are:

Bit 0. The serial port's output buffer is empty and ready to receive another character. This is called a *transmit buffer empty* (TBE) interrupt.

Bit 1. The floppy disk controller's DMA finishes transfering data to or from disk. This is called a *disk block done* interrupt.

Bit 2. A program has set bit 2 in the INTREQ register ($DFF09C). Although any of the hardware interrupts may be falsely triggered through software, this bit is

specifically reserved for software-generated interrupts. To produce friendly, Exec-compatible programs, use the exec.library function Cause( ) to initiate software interrupts.

Bit 3. The Amiga's CIA-A chip generates one of its own interrupts or pin 19 on the expansion bus is set to logic 0. The CIA-A (the first of two Complex Interface Adapters, which takes care of such things as the floppy disk controller, mouse/joystick ports, parallel port, and keyboard) can cause a maskable interrupt. This CIA-A interrupt may be the result of one of the chip's two interval timers counting down to 0, the chip's TOD (Time Of Day) clock reaching alarm time, the chip's serial shift register being emptied (when transmitting data) or being full (when receiving data), or when the chip's FLAG line is pulled low. See the CIA-A's interrupt control register ($BFED01) for more information.

Bit 3 also specifies the enabled/disabled status of external interrupts generated by a card or device connected to the Amiga's expansion bus. This external interrupt occurs when pin 19 on the expansion bus becomes a logic 0 (carries positive voltage).

Bit 4. The copper list sets bit 4 in the INTREQ register. With the copper's specialized instruction set, you can cause an interrupt when the video beam reaches a specific position on the screen. This is sometimes referred to as a *raster interrupt*. To generate a raster interrupt at video line 100, for example, you might use the following copper list instructions:

```
WAIT 0,100
MOVE #$8010,INTREQ
```

Of course, a copper list may cause any of the 14 maskable interrupts simply by setting the appropriate bit 0-13 in the INTREQ register. However, the designers of the Amiga have reserved bit 4 specifically for this purpose.

Bit 5. The video beam reaches the bottom of the screen. When this event occurs, the video beam is turned off and sent back to the top of the screen. This event is referred to as a *vertical blank* because nothing is drawn during this period of time. The computer uses vertical blanks to update its graphics data, thus ensuring a clean, glitch-free picture. On North American NTSC Amigas, this interrupt occurs approximately every 60th of a second. On European PAL systems, vertical blanks occur approximately every 50th of a second. This time interval is often referred to as a *jiffy*.

Bit 6. The blitter finishes an operation. You should always wait for the blitter to complete its current task before you give it something else to do. The graphics.library blitter functions avoid such conflicts for you.

Bits 7-10. An audio channel's DMA outputs the last byte of waveform data. If audio DMA is disabled and your program is manually outputting waveform data

through one of the AUDxDAT registers, audio interrupts occur whenever the AUDxDAT register is empty and ready to receive two more bytes.

Bit 11. The serial port receives a new character. This is called a *receive buffer full* (RBF) interrupt.

Bit 12. The disk controller finds the sync pattern specified by the DSKSYNC register ($DFF07E). AmigaDOS uses the number $4489 as its sync pattern. Under normal operating conditions, disk DMA is activated and subsequent data is read into RAM whenever this interrupt occurs.

Bit 13. The Amiga's CIA-B chip generates one of its own interrupts, or pin 22 on the expansion bus is set to logic 0. The CIA-B (the second of two Complex Interface Adapters which takes care of such things as the serial port, timing for graphics output, and floppy disk and printer control signals) can cause a maskable interrupt. This CIA-B interrupt may be the result of one of the chip's two interval timers counting down to 0, the chip's TOD (Time Of Day) clock reaching alarm time, the chip's serial shift register being emptied (when transmitting data) or being full (when receiving data), or pulling the chip's FLAG line low. See the CIA-B's interrupt control register ($BFDD00) for more information.

Bit 3 also specifies the enabled/disabled status of external interrupts generated by a card or device connected to the Amiga's expansion bus. This external interrupt occurs when pin 22 on the expansion bus becomes a logic 0 (carries positive voltage).

Bit 14. This bit reflects the status of the interrupts' master off switch. If this bit is 0, all maskable interrupts are disabled, regardless of the value of bits 0-13.

$DFF01E INTREQR

Interrupt Request
Status: Read-Only. Chip: Paula

Bit 0: TBE: 1 = A serial transmit buffer empty interrupt has been requested (level 1)
Bit 1: DSKBLK: 1 = A disk block done interrupt has been requested (level 1)
Bit 2: SOFT: 1 = A software interrupt has been requested (level 1)
Bit 3: PORTS: 1 = A CIA-A or expansion port interrupt has occurred (level 2)
Bit 4: COPER: 1 = A copper interrupt has been requested (level 3)
Bit 5: VERTB: 1 = A vertical blank interrupt has been requested (level 3)
Bit 6: BLIT: 1 = A blitter ready interrupt has been requested (level 3)
Bit 7: AUD0: 1 = An audio channel 0 interrupt has been requested (level 4)
Bit 8: AUD1: 1 = An audio channel 1 interrupt has been requested (level 4)
Bit 9: AUD2: 1 = An audio channel 2 interrupt has been requested (level 4)
Bit 10: AUD3: 1 = An audio channel 3 interrupt has been requested (level 4)

Bit 11: RBF: 1 = A serial receive buffer full interrupt has been requested (level 5)
Bit 12: DSKSYN: 1 = A disk sync pattern found interrupt has been requested (level 5)
Bit 13: EXTER: 1 = A CIA-B or an expansion port interrupt has been requested (level 6)
Bit 14: INTEN: 1 = A level 6 interrupt has been requested
Bit 15: SET/CLR: Not used by INTREQR

This register tells you the possible cause of an interrupt. See register INTENAR ($DFF01C) for details on what can cause an interrupt.

Location Range: $DFF020-$DFF026

Disk DMA Registers

Several hardware registers are involved in floppy disk operations. Two CIA-A and CIA-B chip registers provide certain low-level functions: The CIA-A's PRA register ($BFE001) provides disk drive status information, and the CIA-B's PRB register ($BFD100) gives you direct motor and head control as well as allowing you to select which floppy drive is currently active. Further disk status information can be obtained from the DSKBYTR register ($DFF008). DSKBYTR also contains the current byte of data being read from disk. The ADKCON register ($DFF09E) allows you to select the controller's data-storage format (GCR or MFM), move the drive head to a particular data word, and switch between one of four different bit densities for data written to disk. DSKSYNC ($DFF07E) is where you write the word you're looking for when synchronizing the drive head on data.

The following four registers—locations $DFF020-$DFF026—are responsible for all DMA-controlled floppy disk operations. Basically, to initiate a read operation, you tell the computer where the disk data should be stored by placing its addresss in DSKPT, enable disk DMA through the DSKENA bit in the DMACON register, and use the DSKLEN register to initiate read mode, activate disk DMA, and tell the computer how many words of data are to be read. DSKLEN must be written to twice before the command will take effect.

To initiate a write operation, you tell the computer where your data is located by storing its address in DSKPT; enable disk DMA through the DMACON register; and use the DSKLEN register to initiate write mode, activate disk DMA, and tell the computer how many words of data are to be written. Again, the last three steps are all taken care of by the DSKLEN register, which must be written to twice to activate disk DMA.

To find out when a read or write disk operation is complete, you use the disk block

done interrupt, which can be detected through use of the DSKBLK bit (bit 1) in the INTREQR register ($DFF01E).

It's important to note that data is not stored on disk in standard binary format, so it is necessary to code data before writing it to disk and decode data that has been read from disk. The Amiga's dos.library functions do this for you automatically; the hardware does not.

Hardware-wise, the Amiga supports two disk-coding formats: GCR and MFM. AmigaDOS uses MFM format, which fits less data on a disk than GCR but operates at twice the speed. Disk data isn't coded to disguise it or keep it from prying eyes; it's coded so the disk controller will have an easier time reading it. Floppy drives are relatively fragile instruments, prone to speed and alignment problems. Coding data so certain bit patterns that are hard to read occur less often ensures the accuracy of the disk drive. GCR and MFM are simply two tried-and-true methods of storing data to disk. See the description of bit 12 in the ADKCON register ($DFF09E) for more information on the GCR and MFM data formats.

Programming the Amiga's floppy drives entirely through hardware is best left to expert programmers. In fact, many drive operations cannot be handled through hardware unless the operating system is disabled. If you insist on doing it all yourself, please experiment with disks you don't care about. You wouldn't want to destroy important data while beta testing your latest hardware-driven disk routine.

$DFF020 DSKPT ($DFF020 = DSKPTH and $DFF022 = DSKPTL)

Disk Pointer

Status: Write-Only. Chip: Agnus

This is where you store the starting address of your disk data prior to activating disk DMA. DSKPT actually consists two separate hardware registers—DSKPTH (H for the High bits of the address) and DSKPTL (L for the Low bits of the address). Since they're mapped as two consecutive memory locations, it's easiest if your program simply treats them as one 32-bit register.

When writing data to disk, DSKPT specifies the starting address of the data to be written. When reading data from disk, DSKPT specifies where in memory data should be stored. The low bit of this register is always interpreted as 0, so disk data must begin at an even memory location.

$DFF024 DSKLEN
Disk Data Length
Status: Write-Only. Chip: Paula

Bits 0-13: LENGTH: Number of words to read or write
Bit 14: WRITE: 1 = Activate write mode; 0 = Activate read mode
Bit 15: DMAENA: 1 = Activate disk DMA; 0 = Deactivate disk DMA

This register performs three functions: it activates disk DMA, determines whether you're writing to or reading from disk, and specifies the amount of data to be written or read. You must write the same value to this register twice in a row in order for the value to take effect (MOVE #$8100,DSKLEN; MOVE #$8100,DSKLEN, for example). This double-write rule was incorporated to safeguard against accidental disk activity—programs that lose control and start trashing memory can't just shove a number into this register and initiate potentially harmful disk activity.

Bits 0-13. The length of your data in words (number of bytes divided by 2) goes here. Since the length must be stored in just 14-bits, your data cannot be longer than 32,766 bytes. The DMA hardware uses this length value as a counter. Every time a word of data is transferred to or from disk, this counter is decremented and the address found in the DSKPT register is incremented. Transfer stops when the length value reaches zero.

Because of a bug in the Amiga's hardware, you should always specify one more word of data than you actually need transferred. Otherwise, the Amiga neglects to send the last three bits of data when writing to disk and often forgets to send the last byte of data when reading from disk. Specifying one extra word of data keeps this bug from causing any problems.

Bit 14. Set this bit to 1 if you wish to write to disk. When you write to disk, you run the most risk of destroying data. For this reason, it's recommended that you write a $0000 to this register once before disk operations and once after you're finished. This activates read mode and turns off disk DMA.

Bit 15. Setting this bit initiates disk DMA and starts the specified read or write operation. Disk DMA will not start, however, unless the DMAENA bit in the DMACON register ($DFF096) is also set. You'll have to set both of these bits in order to perform any DMA-controlled disk operations. *NEVER* start disk DMA unless the previous disk operation is complete—you could easily trash a disk if you interrupt a disk write operation that's in progress. To find out when a read or write disk operation is complete, use the disk block done interrupt, which can be detected through use of the DSKBLK bit (bit 1) in the INTREQR register ($DFF01E).

$DFF026 DSKDAT

Disk DMA Write

Status: Write-Only. Chip: Paula

This is the data buffer used by disk DMA when transfering data. It's the write address of the DSKDATR register ($DFF008).

$DFF028 REFPTR

Refresh Pointer

Status: Write-Only. Chip: Agnus

The Amiga uses this register as a RAM-refresh address generator—an eight-bit counter responsible for updating the Amiga's dynamic RAM. Its contents are maintained internally by the Agnus chip. This write-only register is accessible for diagnostic purposes only and should never be written to by the programmer. You could easily corrupt the contents of the Amiga's RAM by altering the contents of this register.

Location Range: $DFF02A-$DFF02C

Video Beam Position Write

By writing to these registers, you can force the video beam to move to a particular screen position. The Amiga continually increments this position—faster than you could possibly update these registers—so it's impossible to freeze the beam at any one location. These registers are used for hardware testing purposes.

To the programmer, these registers serve no practical purpose. In fact, writing to VHPOSW can cause software errors, since the Amiga relies on the consistent and timely occurrence of horizontal and vertical blanks for such chores as refreshing dynamic RAM. Changing the beam position disrupts these events.

$DFF02A VPOSW

Veritical Beam Position Write

Status: Write-Only. Chip: Agnus

Bit 0:	V8: The high bit of the vertical beam position (1 = Vertical beam position greater than 255)
Bits 1-14:	Unused
Bit 15:	LOF: Interlace long-frame flag (1 = Long frame; 0 = Short frame)

This is the write address of the VPOSR register ($DFF004).

$DFF02C VHPOSW
Veritical/Horizontal Beam Position Write
Status: Write-Only. Chip: Agnus

Bits 0-7: H1-H8: Video beam's horizontal position specified in increments of two low-resolution pixels
Bits 8-15: V0-V7: Low 8 bits of video beams vertical position

This is the write address of the VHPOSR register ($DFF006).

$DFF02E COPCON
Coprocessor Control
Status: Write-Only. Chip: Agnus

Bit 0: Unused
Bit 1: CDANG: Copper danger bit (1 = Copper can access blitter registers; 0 = Copper cannot access blitter registers)
Bits 2-15: Unused

This one-bit register determines whether or not the copper (coprocessor) has access to the blitter registers. Normally, the copper cannot write to registers that have an address lower than $DFF080. By setting bit 1 of this register, the copper gains access to the blitter registers—registers $DFF040-$DFF07E. (The copper can never write to registers below $DFF040.)

When the computer is first turned on and every time it's reset, this register is cleared. Be sure to set bit 1 if your copper list needs to write to the blitter. For more information on the copper and its instruction set, see locations $DFF080-$DFF08C.

$DFF030 SERDAT
Serial Data Output
Status: Write-Only. Chip: Paula

Bits 0-7: DB0-DB7: Byte of data to be output
Bit 8: DB8 or STP: Ninth data bit or stop bit
Bit 9: STP: Stop bit if sending nine data bits
Bits 10-15: Generally unused

To send data out the serial port, simply store it here. Upon receiving a nonzero value, the contents of this register are moved into the serial-output shift register, a start bit is sent, and your data is shifted out the serial port—low bit first, high bit last—until the last bit has been output and the shift register is empty (its contents equals 0). Since the

start bit is sent automatically, do not include this bit as part of your data. Do, however, provide one or more stop bits (provide only as many stop bits as the receiving computer or device expects—usually just one, sometimes two). The stop bit(s) should immediately follow the highest bit of data you're sending. Because you *must* include at least one stop bit, data written to SERDAT should never equal zero.

As soon as SERDAT's contents are moved into the shift register, the TBE bit (bit 13) in the SERDATR register is set equal to 1, signifying that SERDAT is ready to accept more data. This allows you to queue up another eight or nine data bits while the shift register is still cranking out information. This event can also trigger a level 1 interrupt (see location $DFF01C for more information). When both the shift register and the SERDAT register are empty, SERDATR's TSRE (bit 12) is set.

Note: By convention, only eight or nine data bits (depending on the desired protocol) should be stored in this register at any one time. But because this is a true 16-bit register, up to 15 data bits plus one stop bit can be output in just one shot—an interesting, if questionably useful, feature.

$DFF032 SERPER

Serial Period (Transfer Rate) and Data Bit Control
Status: Write-Only. Chip: Paula

Bits 0-14: RATE: Bit transfer rate specified in increments of 279.4 nanoseconds
Bit 15: LONG: Number of data bits to expect when receiving data
(1 = Nine data bits; 0 = Eight data bits)

The baud rate for the serial port's incoming and outgoing data is determined by the value stored in this register, as is the number of data bits to be expected when receiving data. When receiving data, the Amiga always expects to see one start bit, eight or nine data bits (as determined by bit 15 of this register), and at least one stop bit.

Bits 0-14. The value stored here plus 1 determines how many bus cycles occur between each bit sent or received through the serial port (one bus cycle takes approximately 279.4 nanoseconds). So if one bit is to be sent every *X* bus cycles, you would use the value *X* - 1.

Serial transfer rates are usually specified in *baud rate* (bits per second), not bus cycles. To translate a baud rate into the period value this register expects, use the following formula:

Period = (3579546 / BaudRate) - 1

The period value for 1200 baud, for example, is 2982 because the rounded integer result of (3579546 / 1200) - 1 is 2982. The following table gives the period value for the six most common baud rates:

Baud Rate	Period Value
300	11931
600	5965
1200	2982
2400	1490
4800	745
9600	372

Bit 15. The Amiga can receive data in groups of eight or nine data bits. If this bit is 0, the Amiga's receiving shift register accepts only eight data bits. When set to 1, nine data bits are accepted. This bit has no effect on outgoing data.

$DFF034 POTGO

Pot Port Data
Status: Write-Only. Chip: Paula

Bit 0: START:
1 = Start pot counters
Bits 1-7: Unused (reserved for chip identification)
Bit 8: DATLX: State of pin 5 on game port 1
(1 = Positive voltage; 0 = Zero volts)
Bit 9: OUTLX: Enable voltage output on pin 5 on game port 1
(1 = Output allowed)
Bit 10: DATLY: State of pin 9 on game port 1
(1 = Positive voltage; 0 = Zero volts)
Bit 11: OUTLY: Enable voltage output on pin 9 on game port 1
(1 = Output allowed)
Bit 12: DTARX: State of pin 5 on game port 2
(1 = Positive voltage; 0 = Zero volts)
Bit 13: OUTRX: Enable voltage output on pin 5 on game port 2
(1 = Output allowed)
Bit 14: DATRY: State of pin 9 on game port 2
(1 = Positive voltage; 0 = Zero volts)
Bit 15: OUTRY: Enable voltage output on pin 9 on game port 2
(1 = Output allowed)

This register is instrumental in the reading of proportional input devices such as paddles and proportional joysticks. The position of a proportional input device is returned in registers POT0DAT and POT1DAT ($DFF012 and $DFF014) for game ports 0 and 1, respectively. But before you read POT0DAT or POT1DAT, bit 1 of this register should be set. See locations $DFF012-$DFF016 for a complete description of how this bit affects the POTxDAT registers.

Besides handling the reading of proportional input devices, this register also allows you to output data on pins 5 and 9 of either game port. By default, pins 5 and 9 carry no voltage, but setting the pin's OUTxx and DATxx bits to 1, you can output a pulse of +5 volts. Bits 8 and 9 control pin 5 on game port 1; bits 10 and 11 control pin 9 on game port 1; bits 12 and 13 control pin 5 on game port 1; and bits 14 and 15 control pin 9 on game port 1.

Writing to the DATxx bits has no effect unless the corresponding OUTxx bit is set to 1 at the same time. When both bits are written to with a 1, the specified pin emits a pulse of approximately +5 volts. To output a steady flow of current, these bits must be written to at least every 300 microseconds.

This register can also used to read the current status of the right mouse button. See register POTGOR ($DFF016) for an explanation and an example program.

$DFF036 JOYTEST

JOY0DAT and JOY1DAT Write
Status: Write-Only. Chip: Denise

This register allows you to set the contents of the JOY0DAT and JOY1DAT registers ($DFF00A and $DFF00C). The value stored here affects both registers, allowing you to reset the mouse-position counters for both ports with just one instruction.

Location Range: $DFF038-$DFF03E

Video Strobe Registers

The Agnus chip tracks the position of the video beam with its VPOS and VHPOS registers. This position, however, is not directly available to custom chips Denise and Paula. To inform these chips of important video events such as vertical blanks, four strobe registers are used—STREQU, STRVBL, STRHOR, and STRLONG.

All four strobe registers belong to Denise, with the exception of STRHOR, which belongs to both Denise and Paula. When a strobe register is written to (any value will do), the parent chip knows the video event corresponding to that register has occurred. These strobe registers are very important. Without them, almost every computer operation would be disabled—disk, blitter, audio, graphics, and so on.

To update these registers, Agnus takes advantage of two refresh cycles that occur

before the drawing of each raster line. Either strobe register STREQU, STRVBL, or STRHOR is written to on the first refresh cycle. The following criteria are used to determine which of these three registers are accessed: If the current raster line is within the visible screen area or within the screen's border, Agnus writes to STRHOR. If the current raster line is within the vertical blank area, the computer is in interlace mode *and* a short frame is being drawn (see location $DFF004 for an explanation of long and short frames), Agnus writes to STREQU. If none of the above conditions are true, Agnus writes to STRVBL.

It takes the computer 227.5 bus cycles to draw a raster line. Since bus cycles cannot be split, the computer alternates between 227 and 228 bus cycles. A 228 bus-cycle raster is called a long raster because it takes one more bus cycle to complete. Prior to each long raster, Agnus uses the second refresh cycle to write to strobe register STRLONG. As a result, this register is "strobed" before the drawing of every other raster line.

Do not write to *or* read any of these registers. Doing so would falsely trigger the strobe and severely confuse the Amiga.

$DFF038 STREQU

Short Frame Vertical Blank Strobe
Status: Strobe. Chip: Denise
This strobe register announces raster lines that are within the vertical blank area of a short frame.

$DFF03A STRVBL

Normal Vertical Blank Strobe
Status: Strobe. Chip: Denise
This strobe register announces raster lines that are within the vertical blank area of a long frame or any frame in a noninterlace display.

$DFF03C STRHOR

Horizontal Sync Strobe
Status: Strobe. Chip: Denise/Paula
This strobe register announces raster lines that are within the visible screen area or within the screen's border.

$DFF03E STRLONG

Long Raster Strobe
Status: Strobe. Chip: Denise
This strobe register announces long raster lines that take 228 bus cycles.

Location Range: $DFF040-$DFF074

Blitter Registers

The blitter (short for *block image transfer*) takes care of almost all of the Amiga's bitmapped graphics manipulation, from drawing lines to moving objects and filling outlined areas. Intuition, for example, uses the blitter for such things as drawing gadgets and printing text. And one of the best things about the blitter is that it operates independently, allowing the Amiga to animate objects while the microprocessor is still hard at work executing program instructions.

Copy Mode. The blitter's main task is to copy data from one area of memory to another. This operation is often referred to as a *blit*. The blitter was designed specifically for copying bitmap data, but you can use the blitter to copy whatever type of data you choose.

Data to be copied may come from up to three different locations in memory, but you can have only one destination address. The blitter refers to the three source addresses as source A, source B, and source C. The area of memory to receive the data is called destination D.

When using more than one source address, the blitter must combine the data from the different sources before it can be written back out. To do this, it performs any combination of eight different boolean logic operations called *minterms*. Minterms allow you to AND and OR the bits from source A, B, and C in 256 different ways before they reach their destination. This provides the programmer with great flexibility when copying bitmapped images. Shapes can be inverted, merged, "cookie cut," and so on. Bits 0-7 in the BLTCON0 register ($DFF040) control the minterm settings.

The most common use for the blitter is to move rectangular areas of a bitmapped screen. Because of the way screen bit maps are laid out, it would be impractical if all you could do was copy several consecutive bytes of memory. Instead, you must be able to skip several bytes between each horizontal screen line. This way, you don't have to copy objects that take up the entire width of the screen.

The BLTAMOD-BLTDMOD registers ($DFF060-$DFF066) allow you to specify the difference between the width of the object being copied and the width of the bit map. The BLTSIZE register ($DFF058) defines width and height of the source and destination area. Together these registers allow the blitter to copy any rectangular area of a bitmap.

The memory address for source A, B, and C goes in registers BLTAPT, BLTBPT, and BLTCPT ($DFF50, $DFF04C, and $DFF048), respectively, and the address for destination D goes in register BLTDPT ($DFF54). When specifying the location of source A, B, C and destination D, you must provide the blitter with an even, word-aligned address.

Because bitmapped objects don't always begin on a word aligned address, and their width isn't always an even multiple of 16 (the number of bits in a word), the blitter can mask out (ignore) certain bits in the first and last word of each horizontal line copied from source A. This way, you can copy an area that starts at any pixel location and is of any pixel width you desire. The first-word mask for source A goes in register BLTAFWM ($DFF044). The last-word mask goes in register BLTALWM ($DFF046). In effect, the first word of each line is ANDed with BLTAFWM and the last word of each line is ANDed with BLTALWM before they are written to destination RAM.

To move bitmapped images horizontally with pixel accuracy, it is necessary to shift the image's data one bit at a time. Using bits 12-15 in the BLTCON0 and BLTCON1 registers ($DFF040 and $DFF042), you can tell the blitter to shift source A and/or source B data right by 0-15 bits before it's transferred to its destination. The shifting takes place within the blitter so it does not affect the area of memory actually being copied.

Shifting is not restricted by word boundaries; a word's low bit is shifted into the high bit of the following word. For example, if two consecutive words with binary values of 0110 0011 1010 1111 and 1001 0010 1100 0011 were shifted right three times, the result would be xxx0 1100 0111 0101 and 1111 0010 0101 1000, where xxx are the three low bits from the previous word. If there is no previous word of data—that is, if the current word is the very first word of data being copied—zeros are shifted into the word's high bits.

What happens when you need to shift something left, say by a pixel? Simply tell the blitter to shift your data right by 15 pixels and to copy your data to a location one word left of where you would normally. This is effectively the same as shifting your data left. (Basically, shifting data right 16 - *X* times produces the same bit pattern as shifting it left *X* times. The only difference is that it ends up one word further to the right.) Optionally, you could also operate the blitter in descending mode. In this mode, data is shifted left by the value stored in bits 12-15 of the BLTCON0 and BLTCON1 registers. Here's another reason for using descending mode:

The blitter is usually called upon to move objects around the screen. Generally, a programmer will do this by copying an object's data from a special buffer into the screen's bit map. (The blitter can copy data to only one destination at a time, so you'll need to perform one blitter operation for every bitplane that you wish to affect.) Next, a copy of the screen's background, which is also stored in a buffer, is copied to the screen to erase the object. Finally, the object is copied to a different location on the screen, thus making it move.

In the previous example, data was copied to and from two separate areas of memory. Sometimes it's desirable to copy to and from areas of memory that overlap.

However, this can cause problems because the blitter may copy a piece of data into a memory location that hasn't been read yet, thus destroying whatever used to be there. For example, say memory location 102 holds a $FEDC and memory location 104 holds a $BA98. If you want to copy these two words into locations 104 and 106, the blitter would first take the $FEDC from 102 and move it into 104. Then, when it went to move the word in 104, it would find $FEDC, not $BA98, because that word had been destroyed by the first move operation.

To avoid the problem of overlapping data, the blitter can operate in two modes: ascending and descending. In the previous example, we showed what would happen if you copied data in ascending order, starting at location 102 and moving upward to location 106. If the data was moved in descending order—starting at location 106 and moving downward to location 102—the copy would have worked flawlessly. The rule here is: If the end of the source overlaps the start of the destination, use descending mode. If the start of the source overlaps the end of the destination, use ascending mode. If the source and destination don't overlap at all, use either mode you please. The DESC bit (bit 1) in the BLTCON1 register ($DFF042) determines the blitter's current mode.

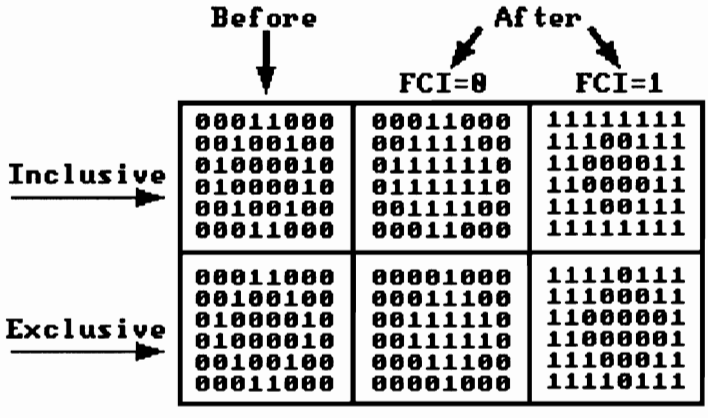
To initiate a blitter copy operation, you should first select copy mode by *not* setting the LINE bit (bit 0) in the BLTCON1 register ($DFF042). Second, tell the computer the address of your source data using the BLTAPT, BLTBPT, BLTCPT register and the address of the destination using the BLTDPT register. (These addresses should specify the starting or ending location of data, depending on whether the blitter is in ascending or descending mode, respectively.) Third, use BLTCON0 to select the desired minterms and tell the computer which sources you're using (A, B, and/or C) and enable the blitter's D destination channel (if you don't enable D, data isn't copied anywhere). Fourth, set up any other options you want, such as shifting or masking of data. Finally, write the width and height of the area to be copied into the BLTSIZE register ($DFF058). Writing to BLTSIZE automatically starts the blitter, so the size should be the last thing you set. During the blitter operation, the BBUSY bit (bit 14) in the DMACONR register is set equal to 1. When the blitter is finished, this bit is cleared.

The following happens during a blitter copy operation: Data is copied one word at a time from the memory specified by the BLTxPT registers to the address found in BLTDPT. Every time a word is copied, the BLTxPT registers are incremented (or decremented if the blitter is in descending mode). When the number of words copied matches the width defined by the BLTSIZE register, the modulation value found in the BLTxMOD registers is added to the BLTxPT registers and the copying continues on the next line. When the number of lines copied equals the height defined by BLTSIZE, the operation is complete and the blitter stops.

Fill Mode. The blitter offers a crude area-fill option that works in tandem with the

data-copy operation described above. Fill mode is activated by setting BLTCON1's IFE (Inclusive Fill Enable) bit or EFE (Exclusive Fill Enable) bit prior to turning on the blitter. After the blitter reads data from memory and manipulates it according to the minterm settings, the specified fill operation is performed on the data before it is written to destination memory. Fill operations work in descending mode only. Figure 3-2 illustrates the effect each type of fill operation has on data:

Figure 3-2. Inclusive and Exclusive Fills



In all fills, the blitter evaluates data one bit at a time, starting at the rightmost edge and moving to the left. Before the fill begins, however, the blitter takes note of the FCI bit (bit 2) in the BLTCON1 register. The value of this bit becomes the initial value of the blitter's fill bit. For the sake of clarity, lets assume that the FCI bit starts out equal to 0.

Here's how an inclusive fill works: As the blitter moves left, it changes all 0 bits to 0, because 0 is the current value of the fill bit. When the blitter encounters a 1 bit, it toggles the fill bit, changing it to a 1. Now the blitter changes following 0 bits to 1, because that is the current value of the fill bit. Every time the blitter encounters a 1 bit in the data, the fill bit changes value. Figure 3-2 illustrates the difference between fills that start with the FCI bit equal to 0 and fills that start with the FCI bit equal to 1.

Exclusive fills take one extra step when they encounter a 1 bit. In addition to toggling the fill bit, the blitter replaces that 1 bit with the new fill-bit value.

Both types of fills expect borders to be defined by a single 1 bit—one bit to tell the blitter it's inside an area, the other to tell it that it's outside the area. When the blitter encounters two 1 bits in a row, it assumes that there is no space to be filled since there is no 0 bit between the two pixels. If the blitter encounters three 1 bits in a row, the blitter

assumes that it has gone in, out, and then back into an area to be filled (the fill bit is toggled three times). This happens whenever the blitter runs across are an odd number of adjacent 1 bits.

To execute a fill operation, simply set up the blitter registers for a standard copy operation, set either the IFE or EFE bit (for an inclusive or exclusive fill, respectively), set the FCI bit to the desired value, set the DESC to put the blitter in descending mode, then start the blitter by writing the area size to the BLTSIZE register. By setting the blitter's source address equal to the destination address, you can fill an area without having to copy data from one separate location to another.

Line Mode. Besides copying data and filling areas, the blitter has one more talent: drawing lines. In line mode, almost all of the blitter registers change their function.

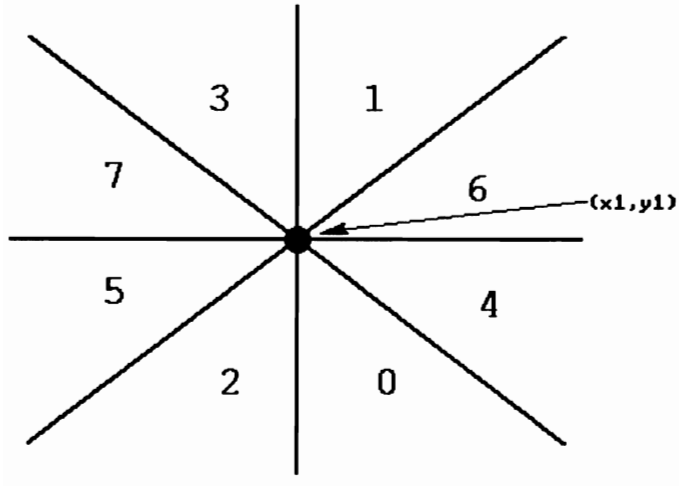
The blitter draws lines at incredible speeds; much faster than the 68000. Unfortunately, you can't just give the blitter two points and then tell it to connect the dots. You need to perform some calculations first.

Let's say that you want to draw a line from point x1,y1 to point x2,y2. From these coordinates you need to figure out the horizontal and vertical distance between the line's two end points. This is easily calculated using the following two formulas:

dx = abs(x1 - x2)
dy = abs(y1 - y2)

Now you're ready to give the BLTCON1 register ($DFF042) some information about the physical orientation of your line. If dx is greater than or equal to dy, and x1 is greater than or equal to x2, set bit 2 equal to 1; if dx is less than dy, and y1 is greater than or equal to y2, set bit 2 equal to 1; if dx is greater than or equal to dy, and y1 is greater than or equal to y2, set bit 3 equal to 1; if dx is less than dy, and x1 is greater than or equal to x2, then set bit 3 equal to 1; finally, if dx is greater than or equal to dy, set bit 4 equal to 1. Together, these bits define the *octant* (position relative to the line's starting point) in which the line will appear. Figure 3-3 shows how the Amiga divides the screen into octants:

Figure 3-3: Octants for Blitter Line Mode



The numbers shown in this figure represent the combined value of BLTCON1's bits 2-4. If a line appears on the border of two octants (as in the case of a straight horizontal line), it does not matter which of the two neighboring octants are selected.

Next you need to determine which value is larger, dx or dy. Let dmax equal the greater value and dmin equal the lesser value: dmax = max(dx, dy); dmin = min(dx, dy). Now, use these values to set the following registers:

BLTBMOD = 4 * dmin
BLTAMOD = 4 * (dmin - dmax)
BLTAPTL = (4 * dmin) - (2 * dmax)

These formulas define the line's slope. If the result of the last calculation—(4 * dmin) - (2 * dmax)—is negative, you must store a 1 in the SIGN bit (bit 6) of the BLTCON1 register.

Besides holding the line's octant number and the negative/positive status of the line's slope, BLTCON1 affects the line's physical appearance. If you're drawing lines to enclose an area that you plan to fill later using blitter fill mode, you should set the ONEDOT bit (bit 1) equal to 1. This tells the blitter to draw lines using only one pixel per raster, thus providing a single-pixel border for your object.

To create textured lines, BLTCON1's bits 12-15 work in conjunction with the BLTBDAT register ($DFF072). The bit pattern found in BLTBDAT defines the pixel

pattern used to draw lines. For normal solid lines, set all of BLTBDAT's bits to 1 ($FFFF). Other values create dotted or dashed lines. Bits 12-15 in BLTCON1 allow you to specify which bit in BLTBDAT, 0-15, defines the status of the first pixel in the line. For most practical purposes, BLTCON1's bits 12-15 should be set equal to the value of x1's lower four bits (x1 AND $0F). This informs the blitter to start the line off with the value found in BLTBDAT's most significant bit, bit 15. *Important: Always set BLTCON1 prior to BLTBDAT.*

BLTCON1's bit 5 should always be set to 0, as should bits 7 through 11. To tell the blitter that you want it to draw lines instead of copy data, the LINE bit (bit 0) must be set to 1.

The Amiga needs certain information about the size and location of the screen's bitmap before it can draw a line. First, store the byte-width (number of pixels divided by eight) of the bitmap in the BLTCMOD and BLTDMOD registers ($DFF060 and $DFF066). Next, you must put the address of the word containing the starting point of the line into the BLTCPT and the BLTDPT registers ($DFF048 and $DFF054).

Only one bitplane can be written to during a single blitter operation. So to draw a line of a particular color on a multiple bitplane screen, it may be necessary to perform two or more line blits. In these cases, you set the blitter registers for the first bitplane, as usual, and perform the blit; then for subsequent bitplanes, you simply re-initialize the registers with the same values, *except* for registers BLTCPT and BLTDPT, which must contain the address of the line's starting point within the new bitplane.

As with blitter copy mode, you must set bits 0-7 in the BLTCON0 register ($DFF040) to choose a minterm. Usually, you should store a $CA here, but if you prefer to XOR your line onto the screen (invert all the pixels found on the line), store a $4A here.

BLTCON0's bits 8-11 should be set equal to $B. (This activates blitter source A and C, and destination D.) Store x1's lower four bits (x1 AND $0F) into BLTCON0's bits 12-15. The blitter uses this value to determine the bit position of the line's starting point within the bitplane memory location specified by registers BLTCPT and BLTDPT.

Now set BLTADAT ($DFF074) equal to $8000 (set this register only *after* you've set BLTCON0). Only two more registers must be set before you can activate the blitter: BLTAFWM and BLTALWM ($DFF044 and $DFF046). Store a $FFFF into both.

Finally, you're ready to start the blitter by writing to the BLTSIZE register ($DFF058). Use the following formula to determine the value that you should store into this register:

BLTSIZE = (dmax * 64) + 66

Because writing to BLTSIZE turns on the blitter, this should be the last register that you set.

General Guidelines. When programming the blitter at the hardware level with multitasking turned on, you must be sure to grab the blitter for your own exclusive use so other programs don't try to use it. Using the library function call OwnBlitter( ) you can reserve the blitter for your own personal use. Function call DisownBlitter( ) frees the blitter so other programs can use it.

Before writing to any of the blitter registers, you must be sure the blitter isn't currently in operation, even after a call to OwnBlitter( ). To ensure the blitter's dormancy, you can use the function call WaitBlit( ), or you can read the status of the BBUSY bit found in the DMACONR register ($DFF002).

Under normal operating conditions, the Amiga's microprocessor has priority over the blitter when it comes to accessing chip RAM. Because of the way memory cycles are allocated, memory conflicts rarely occur between the blitter and the 68000. However, if time is an critical factor, you can give the blitter a higher priority than the 68000 by setting the BLTPRI bit in the DMACON register ($DFF096).

The blitter can perform so many special operations on a word of data that it's important to know the order in which these operations take place. Masking via the BLTAFWM and BLTALWM registers takes effect first (assuming the data was obtained from source A and is either the first or last word of a horizontal line); next the data is shifted, if a shift was specified by the source's shift bits (ASHx or BSHx); the minterm operations are performed next; and finally, the data is filled according to the status of BLTCON1's EFE, IFE, and FCI bits. Only after all this does the data reach its destination.

As a general rule, you should always write 0s to any unused bits in a blitter register. On future versions of the Amiga blitter, these bits may be used to perform new functions—functions that could freak your program if mistakenly activated.

$DFF040 BLTCON0

Blitter Control Register 0
Status: Write-Only. Chip: Agnus

For Blitter Moves and Fills:

Bit 0: LF0: 1 = Selects minterm $\overline{A}\overline{B}\overline{C}$
Bit 1: LF1: 1 = Selects minterm $\overline{A}\overline{B}C$
Bit 2: LF2: 1 = Selects minterm $\overline{A}B\overline{C}$
Bit 3: LF3: 1 = Selects minterm $\overline{A}BC$
Bit 4: LF4: 1 = Selects minterm $A\overline{B}\overline{C}$
Bit 5: LF5: 1 = Selects minterm $A\overline{B}C$

Bit 6: LF6: 1 = Selects minterm AB$\overline{C}$
Bit 7: LF7: 1 = Selects minterm ABC
Bit 8: USED: 1 = Enables DMA for destination D
Bit 9: USEC: 1 = Enables DMA for source C
Bit 10: USEB: 1 = Enables DMA for source B
Bit 11: USEA: 1 = Enables DMA for source A
Bits 12-15: ASH0-ASH3: Preshift value for source A

For Line Mode:
Bits 0-7: LF0-LF7: Minterm value, usually $CA
Bit 8: USED: Set to 1
Bit 9: USEC: Set to 1
Bit 10: USEB: Set to 0
Bit 11: USEA: Set to 1
Bits 12-15: START0-START3: Set to (x1 AND $0F)

Here's how BLTCON0's bits operate in normal copy mode:

Bits 0-7. Each of these bits controls a particular *minterm*—a logical operation that's performed on blitter source data before it reaches its destination. (The bit table above shows the minterms that correspond with each bit.) To activate a minterm, simply set its corresponding bit equal to 1. You can select as many or as few minterms as you please.

Using minterms, you can manipulate bitmapped data in a variety of ways. You can invert images, "cookie cut" images, filter images through a grid (or any kind of pattern, for that matter), merge images, and just plain copy them from one place to another.

The blitter can copy data from three separate sources. Each source is identified by a letter: there's source A, source B, and source C. As words of data are copied from each source, the specified minterm operation is performed on the data to determine what is actually written to memory.

Suppose that only bit 0 of this register is set to 1. In this case, the minterm $\overline{A}\overline{B}\overline{C}$ is selected. A line above a letter means NOT, or not equal to 1; the letters A, B, and C represent blitter sources A, B, and C. So this minterm tells the blitter to output a 1 bit only where there is *not* a 1 bit in source A, source B, and source C. This is the same as if the word from each source was XORed with $FFFF and then ANDed with each other. For example, here's what would result if the blitter copied the following words of data using a minterm setting of $\overline{A}\overline{B}\overline{C}$:

```
Minterm  = ĀB̄C̄
Source A:  1010 1010 1010 1010 = $AAAA
Source B:  1000 1001 1100 1100 = $89CC
Source C:  0010 0110 0100 0000 = $2640
Result:    0101 0000 0001 0001 = $5011
```

Now let's say you set only bit 5 of this register. This selects the minterm AB̄C, which means the blitter outputs a 1 where there is a 1 bit in source A, there is not a 1 bit in source B, and there is a 1 bit in source C. The following shows how this minterm setting would affect the sample data given above:

```
Minterm  = AB̄C
Source A:  1010 1010 1010 1010 = $AAAA
Source B:  1000 1001 1100 1100 = $89CC
Source C:  0010 0110 0100 0000 = $2640
Result:    0010 0010 0000 0000 = $2200
```

You can also calculate the results of this minterm with the formula $AAAA AND ($89CC XOR $FFFF) AND $2640. You see, each source in a minterm is ANDed together to produce the actual result. If the source letter has a line over it, its value is NOTed first (to NOT a number, simply XOR it with $FFFF).

For more complex results, it's possible to combine any of the eight minterms assigned to bits 0-7 of this register. When more than one minterm is selected, the results of those minterms are ORed together to form the final output. For example, writing a 1 to bits 4 and 5 of this register selects the minterm AB̄C̄ + AB̄C, where the plus sign signifies an OR operation. In plain English, this minterm tells the blitter to output a 1 bit where there is a 1 bit in source A and there is not a 1 bit in sources B and C, *or* there is a 1 bit in source A, there is not a 1 bit in source B, and there is a 1 bit in source C. Mathematically, you can think of this as (A AND (B XOR $FFFF) AND (C XOR $FFFF)) OR (A AND (B XOR $FFFF) AND C).

If you look closely at the minterm AB̄C̄ + AB̄C, you'll find that its logic can be simplified. Because the test for source A and B remain the same while the test for source C changes, the value of source C is unimportant—the blitter doesn't care if its value is 0 or 1. As a result, AB̄C̄ + AB̄C can be reduced to AB̄. For the same reason, minterm AB̄C̄ + AB̄C + ABC̄ + ABC can be represented simply as A.

It's usually easiest to determine the minterm that you wish to use in its simplest form first; then expand the minterm into long form so you know exactly which LFx bits to set. For example, if you have a picture defined in source A and you want to output its bits only where the bits in source B correspond, you'd use the logic AB.

Expanding AB is easy—simply select all the minterms that contain the logic AB.

This results in the selection of minterms $AB\overline{C} + ABC$. So to select the minterm setting AB, store a 1 into bits 6 and 7 (write a $C0 into this register's LFx bits).

Now let's complicate things a bit. Say you want to combine the data from source A and source B. For this, you would use the logic A + B (tell the blitter to write a 1 bit if there is a 1 bit in A *or* a 1 bit in B). To expand this, select all the minterms that contain the logic A *or* B in it. This results in the selection of minterms $\overline{A}B\overline{C} + \overline{A}BC + A\overline{B}\overline{C} + A\overline{B}C + AB\overline{C} + ABC$, which means you must set bits 2, 3, 4, 5, 6, and 7. So to use the minterm logic A + B, write a $FC to this register's LFx bits. Using the same process, you could expand the minterm $AB + A\overline{C}$ to $A\overline{B}\overline{C} + AB\overline{C} + ABC$, which would place the value $D0 in the LFx bits.

To fill an area of memory with 0s, don't select any minterms (write a $00 to the LFx bits). Consequently, you can fill an area of memory with 1s by selecting *all* of the minterms (write a $FF to the LFx bits). In both cases, it doesn't matter which sources are activated because their value is, in effect, ignored.

When blitting with only one or two sources active, choose minterms that aren't affected by the value of the unused source(s).

Bit 8. This bit determines whether the blitter's destination DMA is enabled. Setting this bit to 1 enables the DMA; 0 disables it. If destination DMA is disabled, nothing is written to memory during a blit. Being able to operate the blitter without affecting memory can be a useful feature for detecting collisions between two images. Simply set up the blitter for sources A and B to point to two images, select the minterm logic AB ($C0), activate the blitter, wait for the blitter to finish, and then check the BZERO bit (bit 13) of the DMACON register ($DFF002). If BZERO is equal to 0, the images touch. Because the BZERO bit is valid even if data isn't written to memory, you don't need to activate destination D for this type of collision detection to work.

Bit 9-11. These bits determine which of the blitter's sources—A, B, and/or C—are active during a blitter operation. Writing a 1 to any of these bits enables DMA for the corresponding source. When the blitter is turned on, data is read from the memory specified by the active sources' BLTxPT registers. This does not mean, however, that inactive sources have no affect on the data output to destination D. It simply means that the value of that source will not be obtained from memory; instead, the value of an inactive source is always the value that was last written to the source's BLTxDAT register. For this reason, it is important to preinitialize the BLTxDAT register of all unused sources before performing a blit. If you don't want inactive sources to affect the result of a blit, choose minterms that ignore the value such sources.

Bit 12-15. Prior to applying the value of source A to the selected minterm operation, the blitter shifts the value the number of times specified in these bits. (If the blitter is in ascending mode, the data is shifted right. If the blitter is in descending mode, the data is shifted left.) Shifting does not occur if you write a 0 to these bits.

Whenever you tell the blitter to shift the data in source A, you must store a modulo value that's 1 less than normal in the BLTAMOD and BLTDMOD registers.

Line Mode

Here's how the register's bits work in line mode:

Bits 0-7. These bits control the minterm setting used to combine the line's bits (obtained from source A) with the bits in the destination bitplane (obtained from source C). Source B is used as a mask for generating patterned lines. For normal lines, use a minterm value of $CA. To draw a line of inverted pixels, use the minterm value $4A.

Bits 8-11. The blitter draws lines using source A as a point generator, source B as a static mask, source C for reading the current status of the destination bitplane, and destination D for outputting the line's pixels. Because we don't want the mask found in source B to change, source B's DMA is not activated (its bit should be set equal to 0). All of the blitter's other channels—source A, source C, and destination D—must be active, however. As a result, you should always set these bits equal to $B when using the blitter to draw lines.

Bits 12-15. These bits must be set equal to the lower four bits of horizontal coordinate x1 (bit 12 = x1 AND 1; bit 13 = x1 AND 2; bit 14 = x1 AND 4; bit 15 = x1 AND 8). Although these are referred to as START bits in line mode, their contents still specify a shift value, as they do in copy mode. Here's how it works: In line mode, you must initialize source A's data register (BLTADAT) with a $8000. This puts a 1 bit in the left-most position of the word. The value stored in these START bits specifies how far that bit must be shifted to the right. By shifting that bit right by the value found in x1's lower four bits, you move the bit into the pixel position of the line's starting point.

All bits. Here's a quick-and-easy formula for calculating the value that you must store in BLTCON0 when drawing lines:

3018 + ((x1 AND $0F) * 4096)

This specifies the minterm value for a normal line, and provides the blitter with the bit position of x1 within the memory location found in registers BLTCPT and BLTDPT ($DFF048 and $DFF054). To XOR a line onto the screen (invert any bits located within the line), subtract 128 from the result of above calculation.

$DFF042 BLTCON1
Blitter Control Register 1
Status: Write-Only. Chip: Agnus

For Blitter Moves and Fills

Bit 0:	LINE: 1 = Activates line draw mode; 0 = Normal copy mode
Bit 1:	DESC: 1 = Descending operation; 0 = Ascending operation
Bit 2:	FCI: Starting bit value for fill mode
Bit 3:	IFE: 1 = Enables inclusive fill mode
Bit 4:	EFE: 1 = Enables exclusive fill mode
Bits 5-11:	Unused. Always set to 0
Bits 12-15:	BSH0-BSH3: Preshift value for source B

For Line Mode:

Bit 0:	LINE: Set to 1
Bit 1:	ONEDOT: 1 = Draw lines with only a single pixel per raster
Bit 2:	AUL: Set to 1 if ((dx >= dy) and (x1 >= x2)) or ((dx < dy) and (y1 >= y2))
Bit 3:	SUD: Set to 1 if ((dx >= dy) and (y1 >= y2)) or ((dx < dy) and (x1 >= x2))
Bit 4:	SUL: Set to 1 if dx >= dy
Bit 5:	Unused. Always set to 0
Bit 6:	SIGN: Set to 1 if (4 * dmin) < (2 * dmax)
Bits 7-11:	Unused. Always set to 0
Bits 12-15:	TEXTURE0-TEXTURE3: Set to (x1 AND $0F)

Here's how BLTCON1's bits operate in normal copy mode:

Bit 0. This bit activates line mode. It must be set equal to 0 for copy operations.

Bit 1. Storing a 1 in this bit puts the blitter in descending mode. You should use descending mode whenever your source and destination data overlap and the destination data has a higher address than the source. You should also use descending mode if you need to shift your source data left. You must use descending mode if you plan to perform a blitter fill operation.

When descending mode is active, the BLTAPT, BLTBPT, and BLTCPT registers ($DFF050, $DFF04C, and $DFF048) must point to the last word in the memory area to be copied, and the BLTDPT register ($DFF054) must point to the last word in the memory area to be written to. Normally, in ascending mode, these registers point to the start of the memory area.

Bit 2. This value of this bit determines the initial fill value used by the blitter in fill mode. Fill mode is activated by setting either of the following two bits.

Bit 3. Setting this bit equal to 1 activates inclusive fill mode.

Bit 4. Setting this bit equal to 1 activates exclusive fill mode. For normal copy operations, both this bit and bit 3 should be set equal to 0. Fills are performed on data only *after* it has been run through the blitter's current minterm operation.

Bit 12-15. Prior to applying the value of source B to the selected minterm operation, the blitter shifts the value the number of times specified in these bits. (If the blitter is in ascending mode, the data is shifted right. If the blitter is in descending mode, the data is shifted left.) Shifting does not occur if you write a 0 to these bits.

Whenever you tell the blitter to shift the data in source B, you must store a modulo value that's 1 less than normal in the BLTBMOD and BLTDMOD registers.

Line Mode. To activate line mode, set bit 0 of this register equal to 1. Setting the ONEDOT bit (bit 1) tells the blitter to draw lines that have only one pixel on each horizontal row. For clear clean lines, you should set this bit equal to 0. But if you are drawing objects that you plan to fill later using the blitter's fill function, you should set this bit equal to 1. This way, the area to be filled will be enclosed by single-pixel walls, just as the blitter expects them to be. Bit 5 and bits 7 through 11 should all be set to 0. The value of the remaining bits depends on the line's coordinates. (See this register's bit table above.)

You might be interested to know that this register's TEXTURE bits are actually used as a shift value, as they are in blitter copy mode. You see, blitter source B is used as a mask for creating textured lines. Because a line may start at any pixel position, it's necessary to shift the mask found in BLTBDAT to align its high bit with the line's starting point. As with bits 12-15 in the BLTCON0 register, the shift value for source B is calculated from the lower four bits of x1.

$DFF044 BLTAFWM

Source A First Word Mask

Status: Write-Only. Chip: Agnus

This register allows you to mask out the bits found on the left edge of the bitplane area specified for blitter source A. The first word of each horizontal line is ANDed with the value stored in this register before it is shifted, applied to the minterm, and copied to memory. For example, to ensure that the two left-most pixels of an area are always set to 0, store a $3FFF in this register. In descending mode, this and the following register swap functions.

Line Mode. In line mode, you should always store a $FFFF here. Otherwise, your line may appear broken.

$DFF046 BLTALWM

Source A Last Word Mask

Status: Write-Only. Chip: Agnus

This register allows you to mask out the bits found on the right edge of the bitplane area specified for blitter source A. The last word of each horizontal line is ANDed with the value stored in this register before it is shifted, applied to the minterm, and copied to memory. For example, to ensure that the two rightmost pixels of an area are always set to 0, store a $FFFC in this register. In descending mode, this and the previous register swap functions.

Line Mode. In line mode, you should always store a $FFFF here. Otherwise, your line may appear broken.

$DFF048 BLTCPT ($DFF048 = BLTCPTH and $DFF04A = BLTCPTL)

Blitter Source C Pointer

Status: Write-Only. Chip: Agnus

This is where you store the address of source C's bit-map data prior to activating the blitter. If you're blitting in ascending order, this address should specify the starting address of your data. If you're blitting in descending order, this address should specify the ending address of your data—the address of the last word of data.

BLTCPT actually consists two separate hardware registers—BLTCPTH (H for the High bits of the address) and BLTCPTL (L for the Low bits of the address). Since they're mapped as two consecutive memory locations, it's easiest if your program simply treats them as one 32-bit register.

The low bit of this register is always interpreted as 0, so blitter bitmap data must begin at an even memory location (using the BLTxFWM and BLTxLWM mask registers, however, you can specify bitmapped areas with single-pixel accuracy).

Line Mode. This register, as well as the BLTDPT register ($DFF054), must contain the starting address of the line to be drawn—that is, the address of the word that contains the first point in the line. Using standard (*x1,y1*) coordinates where (0,0) is the upper-left corner of the bitplane, you can calculate this address using the following formula:

Address = Bitplane + (y1 * BytesPerLine) + (2 * (x1 / 16))

where **Bitplane** is the starting address of the destination bitplane and **BytesPerLine** is the width of the bitplane in bytes (number of pixels divided by eight). So to draw a line in a 320 by 200 pixel bitplane found in memory at 16384 starting at coordinate (10,20), you would store a 17185 into this register because 16384 + (20 * (320 / 8)) + (2*(10/16)) equals 17185.

$DFF04C BLTBPT ($DFF04C = BLTBPTH and $DFF04E = BLTBPTL)

Blitter Source B Pointer

Status: Write-Only. Chip: Agnus

This is where you store the address of source B's bitmap data prior to activating the blitter. See register BLTCPT ($DFF048) for details.

Line Mode. This register is not used in blitter line mode.

$DFF050 BLTAPT ($DFF050 = BLTAPTH and $DFF052 = BLTAPTL)

Blitter Source A Pointer

Status: Write-Only. Chip: Agnus

This is where you store the address of source A's bitmap data prior to activating the blitter. See register BLTCPT ($DFF048) for details.

Line Mode. Only the low word of this register, BLTAPTL ($DFF052), is used in line mode. Its value determines the slope of the line. Use the following formula to calculate the value of this register:

BLTAPTL = (4 * dmin) - (2 * dmax)

where **dmin** equals min(abs(x1 - x2), abs(y1 - y2)) and **dmax** equals max(abs(x1 - x2), abs(y1 - y2)). If this formula produces a negative number, you must write a 1 to the SIGN bit (bit 6) of the BLTCON1 register ($DFF042).

$DFF054 BLTDPT ($DFF054 = BLTDPTH and $DFF056 = BLTDPTL)

Blitter Destination Pointer

Status: Write-Only. Chip: Agnus

This is where you store the address of the chip RAM you're blitting into (usually, this is a location within a bitmap). If you're blitting in ascending order, this address should reflect the starting address of the destination area. If you're blitting in descending order, this address should reflect the ending address of the destination area. This register has the same limitations as the BLTAPT-BLTCPT registers. See location $DFF048 for details.

Line Draw Mode. This register, as well as the BLTCPT register ($DFF048), must contain the starting address of the line to be drawn—that is, the address of the word that contains the first point in the line. See location $DFF048 for more information.

$DFF058 BLTSIZE

Blitter Start and Size
Status: Write-Only. Chip: Agnus

For Blitter Moves and Fills
Bits 0-5: W0-W5: Width of destination area in words
Bits 6-15: H0-H9: Height of destination area in pixels

For Line Mode:
Bit 0: Set to 0
Bit 1: Set to 1
Bits 2-5: Set to 0
Bits 6-15: Set to (dmax + 1)

The value stored in this register tells the blitter how large your destination area is. The lower six bits specify the width of the area in words—pixels divided by 16. If these bits are set to 0, the blitter assumes a width of 64 words, or 1024 pixels. The height of the area in pixels goes in the upper nine bits. A 0 specifies a height of 1024 pixels. You can use the following formula to calculate the value of this register:

BLTSIZE = (Height * 64) + (PixelWidth / 16)

The maximum width or height is 1024 pixels. The minimum width is 16 pixels and the minimum height is 1 pixel.

Writing to this register activates the blitter. As a result, it should be the last blitter register that you write to.

If you're shifting source A or source B data, you must specify a horizontal width that is 1 greater than normal.

Line Mode. In line mode, the lower six bits must hold the value 2 while the upper 9 bits must hold the value of dmax plus 1. Use the following formula to determine the value that you should store into this register:

BLTSIZE = (dmax * 64) + 66

where **dmax** equals max(abs(x1 - x2), abs(y1 - y2)).

Because writing to BLTSIZE turns on the blitter, this should be the last register you set.

Location Range: $DFF05A-$DFF05E

Unused

$DFF060 BLTCMOD

Blitter Source C Modulo

Status: Write-Only. Chip: Agnus

Whenever the blitter finishes writing data to the last word in a horizontal line, the contents of this register are added to the BLTCPT register so that it points to the start of the next line. This allows the blitter to copy areas within a bitmap that are smaller in width than the bitmap itself. If you're copying an area that is the same width of the bitplane it is contained in, or you are copying consecutive bytes of memory, simply write a 0 to this register. Otherwise, this register should contain the difference between the byte width of the bitplane containing the area being copied and the byte width of that area. Here's a formula for determining the value that should be stored in this register:

BLTCMOD = (BitMapWidth / 8) - (2 * (AreaWidth / 16))

where **BitMapWidth** and **AreaWidth** are the pixel widths of the source bitmap and area to be copied, respectively. Note that this register specifies width in bytes whereas the BLTSIZE register specifies width in words. Also, although this register reflects the bit map width in bytes, its low bit is ignored, so the value here must specify an even number of bytes.

With a separate BLTxMOD register for each source, it's possible to combine the data from three bit maps of different widths.

Line Mode. For drawing lines, store the byte width (pixel width divided by 8) of your bitmap in this register.

$DFF062 BLTBMOD

Blitter Source B Modulo

Status: Write-Only. Chip: Agnus

Whenever the blitter finishes writing data to the last word in a horizontal line, the contents of this register are added to the BLTBPT register so that it points to the start of the next line. See the BLTCMOD register ($DFF060) for details on how to use this register in copy mode.

Line Mode. This register should contain the value **4 * dmin,** where **dmin** equals min(abs(x1 - x2), abs(y1 - y2)).

$DFF064 BLTAMOD

Blitter Source A Modulo

Status: Write-Only. Chip: Agnus

Whenever the blitter finishes writing data to the last word in a horizontal line, the contents of this register are added to the BLTAPT register so that it points to the start of the next line. See the BLTCMOD register ($DFF060) for details on how to use this register in copy mode.

Line Mode. This register should contain the value **4 * (dmin - dmax),** where **dmin** equals min(abs(x1 - x2), abs(y1 - y2)) and **dmax** equals max(abs(x1 - x2), abs(y1 - y2)).

$DFF066 BLTDMOD

Blitter Destination Modulo

Status: Write-Only. Chip: Agnus

Whenever the blitter finishes writing data to the last word in a horizontal line, the contents of this register are added to the BLTDPT register so that it points to the start of the next line. See the BLTCMOD register ($DFF060) for details on how to use this register in copy mode.

Line Mode. For drawing lines, store the *byte width* (pixel width divided by 8) of your bitmap in this register. This is the same value you must store in the BLTCMOD register.

Location Range: $DFF068-$DFF06E

Unused

Location Range: $DFF070-$DFF074

Blitter Data Registers

The blitter uses these registers as a temporary holding place for data that is read from memory. If a source's DMA is enabled for a blitter operation, words are read from memory and then stored in the corresponding BLTxDAT register. Here the data is manipulated according to the mask registers, the shift bits, the minterm operation, and then it is written to destination RAM.

If a source's DMA is disabled, data is not read from memory, but the value found in the source's data register is still used to form the blitter's output. For this reason, it's a good idea to initialize the data register of an unused source prior to starting the blitter.

You can use data registers to fill an area of memory with any word-long value. Simply set up the blitter so that none of the data registers have DMA enabled, select a

minterm of A, write the desired value into the BLTADAT register, set up the remaining registers as usual, and activate the blitter.

If you plan to shift source A or source B data during a blitter operation, you must be sure to initialize the shift bits prior to writing to the source's data register. Otherwise, the number written to this register will be shifted according to the old shift value.

$DFF070 BLTCDAT

Blitter Source C Data
Status: Write-Only. Chip: Agnus
This is the data register for blitter source C.

Line Mode. This register is not used in line mode.

$DFF072 BLTBDAT

Blitter Source B Data
Status: Write-Only. Chip: Agnus
This is the data register for blitter source B.

Line Mode. Store the bit pattern with which you wish your line to be drawn in this register. For normal solid lines, store a $FFFF here. Other values produced dotted or dashed lines.

$DFF074 BLTADAT

Blitter Source A Data
Status: Write-Only. Chip: Agnus
This is the data register for blitter source A.

Line Mode. This register should be initialized with the value $8000 for line mode.

Location Range: $DFF076-$DFF07C

Unused

$DFF07E DSKSYNC

Disk Sync Pattern
Status: Write-Only. Chip: Paula
Before reading data from disk, it's often necessary to synchronize the drive's head on a particular bit pattern. This register allows you to do that just that.

When the WORDSYNC bit (bit 10) in the ADKCON register ($DFF09E) is set, the disk controller's DMA is enabled and the controller prepares to search the disk for

the sync pattern found in this register. The disk controller doesn't start searching until this register is written to. When the sync pattern is found, subsequent data is read into RAM. Bit 12 of the DSKBYTR register ($DFF01A) is set to 1 for two or four microseconds (depending on the setting of ADKCON's bit 8) as soon as the sync pattern is located. This event can also be used to trigger a level 6 interrupt.

In MFM format (the disk format used by AmigaDOS), the sync pattern should be a value that is impossible to create using MFM data coding. This way, it can't be confused with actual data. AmigaDOS uses the number $4489 as its sync pattern.

For more information on low-level disk access, see CIA-B's PRB register at $BFD100.

Location Range: $DFF080-$DFF08C

Copper Registers

These registers control the copper. *Copper* is short for coprocessor. It is a microprocessor dedicated to controlling the Amiga's video system. The copper is designed to free the 680x0 from display tasks.

The copper runs programs called *copper lists*. A copper list consists of several copper instructions. Normally, the operating system's View routines create and manage copper lists transparently. If you're programming productivity or utility software, you should not tamper with the system's copper list directly. However, there are operating system calls—like CWait( ), CMove( ), and CBump( )—which can be used if you'd like to make an Intuition-friendly color change (or other alteration) mid-screen.

If you've thrown out the operating system (for an arcade-style game, most likely), you can build your own copper lists from scratch. A helpful exercise is to use a debugger to look at the system's Workbench copper list (be sure you're not looking at the debugger's copper list). Or, you can write a program to do the snooping. You can find the address of the system's copper list in the ViewLord element in the IntuitionBase structure. But to "disassemble" the copper list, you'll need to learn about the copper's instruction set.

There are three copper instructions: WAIT, MOVE, and SKIP. It doesn't seem like you could do much with only three instructions, but that's not the case. Using a variety of tricks and subterfuges, a copper list can perform loops, take over the blitter, and even interrupt the 68000 for special processing.

WAIT. WAIT mimics a loop structure which waits for a certain condition to occur. Specifically, WAIT waits for the *x* and *y* coordinates of the video beam to meet or exceed the *x* and *y* coordinates which you specify. Thus, WAIT allows you to synchronize actions with the video. For example, you could change color register 0 (the

background color) halfway down the screen. Since a typical copper list executes every time the screen is drawn, the color change will be rock-steady.

The *x* and *y* coordinates are not simple pixel coordinates. Instead, they correspond to the the values of the beam position counter. See the descriptions of the registers VPOSR ($DFF004), VHPOSR ($DFF006), VPOSW ($DFF02A), and VHPOSW ($DFF02C).

Here is what a WAIT instruction looks like:

Bit 0:	always set FALSE (0)
Bits 1-7:	horizontal compare enable bits
Bits 8-14:	vertical compare enable bits
Bit 15:	the blitter-finished-disable bit (usually 1)
Bit 16:	always set TRUE (1)
Bits 17-23:	horizontal beam position
Bits 24-31:	vertical beam position

Bits 1-7 describe which bits of the horizontal beam counter will be compared with a *x* position for which you choose to wait. Normally, you should set all these bits to 1.

Bits 8-14 describe which bits of the vertical beam counter will be compared with a *y* position for which you choose to wait. Normally, you should set all these bits to 1.

Bit 15 should normally be set to 1. If you set it to 0, the copper WAIT instruction will wait until the beam position matches or exceeds the given position and the blitter has finished its operation. Frankly, it's hard to imagine why anyone would want to set this bit to 0.

Bits 17-23 indicate the horizontal beam position for which the WAIT is waiting. Note that bits 1-7 are used to decide which bits are actually used in the comparison.

Bits 24-31 indicate the vertical beam position for which the WAIT is waiting. Note that bits 8-14 are used to decide which bits are actually used in the comparison.

Because the horizontal beam position is only maintained to a resolution of two low resolution pixels, and because the least significant bit of that position is not used in the WAIT comparison, WAIT has a resolution of four low resolution or eight high resolution pixels.

One potential problem you may run into with WAIT is that you cannot tell the difference between a vertical line *y* and its counterpart y+256. For example, vertical line 0 looks like vertical line 256, 1 looks like 257, and so on. To wait for a line greater than 255, preceed the WAIT with a WAIT for vertical line 255.

A version of the WAIT command is used to end copper lists. The command code $FFFFFFFE (which waits for a physical location that is guaranteed not to exist) means wait for line 255, horizontal position 254. The loop seems to be infinite, and it is, but

the operating system resets the copper's instruction counter during every vertical blanking period.

MOVE. The MOVE instruction transfers data to a hardware register. The instruction can store data into any register with an address of $80 or more. Note that the register base ($DFF000) is implicit—it is not included in the instruction. If the copper danger bit (CDANG) is TRUE (that is, if bit 1 of COPCON ($DFF02E) is set to 1), the instruction can store data into any register with an address of $40 or more. Unless you need to have the copper write to the blitter registers, you should keep CDANG set to 0.

With the MOVE instruction, you can set up bitplane pointers, sprite pointers, color registers, and so on. A complete, useful copper list will have many such MOVE instructions, because there are several DMA registers that need to be reset for each screen frame. For example, the bitplane registers must be reset to keep them from continuing from where they left off. Otherwise, you'd see all of chip memory rushing through the screen at great speed.

Keep in mind that MOVE is not instantaneous. Two MOVE instructions will be seperated by four low resolution pixels.

Here is the format of the MOVE command:

Bits 0-15:	Data
Bit 16:	Set to FALSE (0)
Bits 17-24:	Register address
Bits 25-31:	Unused—set to 0

Bits 0-15 are a word of data you wish to transfer into a hardware register. Since only word values can be placed into a register, it takes two MOVES to set register pairs like BPL1PTH and BPL1PTL, which together make up BPL1PT. The command codes $00E00001,$00E22000 perform two MOVE instructions, which place the address $12000 into BPL1PT, the first bitplane pointer.

Bits 17-24 specify the register address (from the offset of $DFF000), which you would like to modify. Note that only one word will be modified. The CDANG bit of the COPCON register affects which values are considered legal by the copper.

SKIP. Most copper lists are composed of only WAIT and MOVE commands. The SKIP command allows for conditional branching. It is a command that has quite a bit of potential which has rarely been tapped.

SKIP is a close relative of WAIT. As is the case with WAIT, SKIP decides what to do based on the position of the video beam. Instead of waiting for a beam position though, it alters copper list program flow based on the beam position.

The SKIP instruction looks like this:

Bit 0: Set to TRUE (1)
Bits 1-7: Horizontal compare enable bits
Bits 8-14: Vertical compare enable bits
Bit 15: Blitter-finished-disable bit
Bit 16: Set to TRUE (1)
Bits 17-23: Horizontal beam position
Bits 14-31: Vertical position

For more information on these bits, see the WAIT instruction description.

The SKIP command skips the instruction that immediately follows it if the beam counter meets or exceeds the value you specify. The instruction that follows SKIP is typically a strobe to the COPJMP1 or COPJMP2 registers. Writing to these register causes a jump to the address in COP1LC or COP2LC registers, respectively.

Here is a sample copper list. This is a typical NTSC Workbench screen.

Hex code	**Psuedo-code**	**Comment**
		; Wait for correct position
2B01FFFE	WAIT y=43,x=0	
		; Set the playfield color registers.
		; These are controlled by Preferences.
01800000	MOVE 180,000	Color register 0
01820BBB	MOVE 182,BBB	Color register 1
01840009	MOVE 184,009	Color register 2
01860F44	MOVE 186,F44	Color register 3
		; Set the mouse pointer color registers.
		; These are controlled by Preferences.
01A00000	MOVE 1A0,000	Color register 16
01A20D22	MOVE 1A2,D22	Color register 17
01A40000	MOVE 1A4,000	Color register 18
01A60ABC	MOVE 1A6,ABC	Color register 19
		; Set the remaining sprite colors.
		; These are default.
01A80444	MOVE 1A8,444	Color register 20
01AA0555	MOVE 1AA,555	Color register 21
01AC0666	MOVE 1AC,666	Color register 22
01AE0777	MOVE 1AE,777	Color register 23
01B00888	MOVE 1B0,888	Color register 24
01B20999	MOVE 1B2,999	Color register 25
01B40AAA	MOVE 1B4,AAA	Color register 26
01B60BBB	MOVE 1B6,BBB	Color register 27
01B80CCC	MOVE 1B8,CCC	Color register 28
01BA0DDD	MOVE 1BA,DDD	Color reguster 29
01BC0EEE	MOVE 1BC,EEE	Color register 30

Hex code	Psuedo-code	Comment
01BE0FFF	MOVE 1BE,FFF	Color register 31
		; Set the required video registers.
008E0581	MOVE 08E,0581	DIWSTRT
01000200	MOVE 100,0200	BPLCON0
01040024	MOVE 104,0024	BPLCON2
009040C1	MOVE 090,401C	DIWSTOP
0092003C	MOVE 092,003C	DDFSTRT
009400D0	MOVE 094,00D0	DDFSTOP
01020000	MOVE 102,0000	BPLCON1
01080000	MOVE 108,0000	BPL1MOD
010A0000	MOVE 10A,0000	BPL2MOD
00E00000	MOVE 0E0,0000	BPL1PTH
00E2AD78	MOVE 0E2,AD78	BPL1PTL
00E40000	MOVE 0E4,0000	BPL2PTH
00E6EBF8	MOVE 0E6,EBF8	BPL2PTL
		; Wait for correct position, then turn on display
2C01FFFE	WAIT x=0,y=59	
0100A200	MOVE 100,A200	BPLCON0
		; Wait for correct position, then turn off display
F401FFFE	WAIT x=0,y=244	
01000200	MOVE 100,0200	BPLCON0
		; Wait forever
FFFFFFFE	WAIT x=254,y=255	

Depending on memory configuration and your Preferences settings, some of these numbers may be different if you check the copper list of your own system. Notice that there are no sprite commands in this copper list. The Amiga's operating system updates the sprite pointers during vertical blank.

$DFF080 COP1LC ($DFF080 = COP1LCH and $DFF082 = COP1LCL)

Copper Program Counter 1

Status: Write-Only. Chip: Agnus

This is the first copper location register. It is loaded into the copper's program counter whenever COPJMP1 ($DFF088) is written to. The low bit is ignored; as a result this must point to an even (word-aligned) address.

$DFF084 COP2LC ($DFF084 = COP2LCH and $DFF086 = COP2LCL)

Copper Program Counter 2

Status: Write-Only. Chip: Agnus

This is the second copper location register. It is loaded into the copper's program

counter whenever COPJMP2 ($DFF08A) is written to. The low bit is ignored; as a result this must point to an even (word-aligned) address.

$DFF088 COPJMP1

Copper Jump Strobe 1
Status: Strobe. Chip: Agnus
This location, when written to, causes the value previously stored in COP1LC ($DFF080) to be moved to the copper's program counter. The operating system strobes this line during a vertical blank interrupt.

$DFF08A COPJMP2

Copper Jump Strobe 2
Status: Strobe. Chip: Agnus
This location, when written to, causes the value previously stored in COP2LC ($DFF084) to be moved to the copper's program counter.

$DFF08C COPINS

Copper Instruction Identity
Status: Write Only. Chip: Agnus
This location is of questionable use to the programmer. It is written to by the hardware. It is used to identify the current copper instruction.

Location Range: $DFF08E-$DFF094

Position of Screen Display
These four registers define the position of the Amiga's screen. DIWSTRT (Display Window Start) defines the screen's upper-left corner while DIWSTOP (Display Window Stop) defines the screen's lower-right corner. Anything outside of this area is the screen's border.

Before explaining how the next two registers, DDFSTRT and DDFSTOP, affect the display, a brief explanation of bitplane DMA is in order. Bitplane data describes what to draw, but it's the responsibility of bitplane DMA to grab that data from chip RAM and to give it to the video hardware via registers BPL1DAT-BPL6DAT ($DFF110-$DFF11A). Shortly after the video hardware receives new data, it draws that data at the current video beam position. So *where* graphics appear onscreen is determined by *when* bitplane DMA gives the video hardware its information. (Anytime bitplane DMA is inactive and the BPLxDAT registers are empty, the Amiga displays the color specified by the COLOR00 register.)

Vertically, bitplane DMA starts the moment it reaches the Y coordinate found in DIWSTRT and stops once it hits the Y coordinate found in DIWSTOP. Horizontal start and stop locations are a bit more complicated, however, as the computer needs some time before it can display fetched data. With a high-resolution display, it takes the video hardware 4.5 color clocks to digest each group of DMA-fetched data. On a low-resolution display, it takes 8.5 color clocks. (A *color clock* is equal to one memory access cycle, which is approximately 280 nanoseconds in duration. The Amiga can output two low-resolution pixels per color clock.)

Each horizontal line on the computer screen can be thought of as a time line. Where along this line the bitplane DMA is allowed to start is determined by the DDFSTRT (Display Data Fetch Start) register. The DDFSTOP (Display Data Fetch Stop) register tells the Amiga when the bitplane DMA should stop. As you can imagine, unless these registers correspond with the horizontal coordinates found in the DIW registers, your display will appear cropped and/or garbled.

Finding the proper value for DDFSTRT based on the horizontal coordinate found in DIWSTRT is easy. Because the DIWSTRT register specifies coordinates in low-resolution pixels and the DDFSTRT register is accurate to only two low-resolution pixels, you must first divide DIWSTRT's horizontal coordinate by 2; then, simply subtract the number of color clocks you need—4.5 for high-resolution screens, and 8.5 for low-resolution screens. Here are the formulas you should use to calculate the correct value for the DDFSTRT register:

DDFSTRT = HSTART / 2 - 4.5

for high-resolution screens and

DDFSTRT = HSTART / 2 - 8.5

for low-resolution screens. **HSTART** refers to bits 0-7 of the DIWSTRT register (its horizontal coordinate).

Since the default value for HSTART is $81, DDFSTRT defaults to $3C for high-resolution screens and $38 for low-resolution screens.

Calculating the value for DDFSTOP is also easy. Its value depends on the screen's width:

DDFSTOP = DDFSTRT + (4 * ((Width / 16) - 1))

for high-resolution screens and

DDFSTOP = DDFSTRT + (8 * ((Width / 16) - 1))

for low-resolution screens. **Width** is the width of the screen in pixels.

So on a standard 640-pixel high-resolution screen where DDFSTRT equals $3C, DDFSTOP should be set to $D4.

$DFF08E DIWSTRT

Display Window Start
Status: Write-Only. Chip: Agnus

Bits 0-7: HSTART: Horizontal coordinate of screen's upper-left corner

Bits 8-15: VSTART: Vertical coordinate of screen's upper-left corner

Bits 0-7. These bits hold the horizontal coordinate of the upper-left corner of the screen. This value defaults to $81 on most systems. Unlike the VHPOS register, the horizontal position held in these bits is specified in low-resolution pixels; VHPOS specifies horizontal screen positions with an accuracy of only two low-resolution pixels. Figure 3-1 (found near the description for the VPOS register, location $DFF006) shows how both the vertical and horizontal positions returned by VPOSR and VHPOSR relate to a standard NTSC video display. You can use this figure to locate a desired horizontal starting position, but you must remember to multiply that value by 2 to convert it to the resolution that this register expects.

Bits 8-15. These bits hold the vertical coordinate of the upper-left corner of the screen. It's value defaults to $2C. The vertical position is always specified in noninterlace lines, no matter what mode the computer is in. The value stored here corresponds directly with the value returned by the VPOSR and VHPOSR registers ($DFF004-$DFF006). See Figure 3-1 to see how this value relates to the screen.

Although you can't read the contents of the DIWSTRT register, Intuition keeps track of the starting horizontal (HSTART) and vertical (VSTART) positions of the visible screen area in the DxOffset and DyOffset elements of IntuitionBase's ViewLord View structure, respectively.

$DFF090 DIWSTOP

Display Window Stop
Status: Write-Only. Chip: Agnus

Bits 0-7: HSTOP: Horizontal (X) coordinate of screen's lower-right corner

Bits 8-15: VSTOP: Vertical (Y) coordinate of screen's lower-right corner

Bits 0-7. The minimum horizontal stop (HSTOP) position is $100, which is about 1/3 of the way across the screen. The value stored in these bits represents an offset from this position. So the actual horizontal coordinate of the lower-right corner of the screen is $100 plus the value stored here. You can think of HSTOP as a nine-bit value with lower eight bits (bits 0-7) that are specified here and with ninth bit (bit 8) that is always equal to 1. The default value for these bits is $C1, which specifies an HSTOP position of $1C1.

You can use Figure 3-1 to find an HSTOP position, but the horizontal value shown in this figure must be multiplied by 2 (to convert it to the resolution used by this register) and ANDed with $FF (to find the coordinate's lower eight bits).

Bits 8-15. These bits hold the vertical coordinate of the lower-right corner of the screen (the VSTOP position). The minimum VSTOP position is $80. If the value of these bits is less than $80, VSTOP equals $100 plus the value found here. Otherwise, the coordinate found in these bits is taken as is.

Like HSTOP, you can think of VSTOP as a nine-bit value with lower eight bits specified here. Unlike HSTOP, however, VSTOP's ninth bit is not always equal to 1; instead, its value is always the opposite of the eighth bit (bit 15 of this register). If the eighth bit is 0, the ninth bit equals 1. If the eighth bit is 1, the ninth bit equals 0.

The default value for these bits is $F4, which, of course, specifies a VSTOP position of $F4. (The default value for PAL systems is $2C, which specifies a VSTOP of $12C.) VSTOP always specifies the vertical position in noninterlace lines, no matter what mode the computer is in.

$DFF092 DDFSTRT

Display Data Fetch Start

Status: Write-Only. Chip: Agnus

This register specifies the horizontal position that the video beam must reach in order for bitplane DMA to start. It's value depends on your screen resolution and what you store in the HSTART bits (bits 0-7) in the DIWSTRT register ($DFF09E).

In high-resolution mode, only bits 2-7 of this register are active. No matter what you store here, the lower two bits are interpreted as 0s. This keeps the data fetch start value an even multiple of four. In low-resolution mode, only bits 3-7 are active, and the lower three bits are interpreted as 0s. This keeps the data fetch start value an even multiple of eight.

Because of certain hardware limitations, do not use DDFSTRT values less than $18. And because sprite DMA relies on color clocks that occur during raster positions $16-$34, using DDFSTRT values below $38 will disable certain sprites. The lower the number, the more sprites are disabled. Higher numbered sprites are disabled first.

$DFF094 DDFSTOP

Display Data Fetch Stop

Status: Write-Only. Chip: Agnus

This register specifies the horizontal position the video beam must reach in order for bitplane DMA to stop. It's value depends on the how long you want each raster line to be and what you store in the DDFSTRT register ($DFF092).

In high-resolution mode, only bits 2-7 of this register are active. No matter what you store here, the lower two bits are interpreted as 0s. This keeps the data fetch stop value an even multiple of four. In low-resolution mode, only bits 3-7 are active, and the lower three bits are interpreted as 0s. This keeps the data fetch stop value an even multiple of eight.

Because of certain hardware limitations, do not use values greater than $D8.

$DFF096 DMACON

DMA Control

Status: Write-Only. Chip: Agnus/Denise/Paula

Bit 0: AUD0EN: 1 = Enable audio channel 0 DMA (start sound); 0 = Disable audio channel 0 DMA (stop sound)

Bit 1: AUD1EN: 1 = Enable audio channel 1 DMA (start sound); 0 = Disable audio channel 1 DMA (stop sound)

Bit 2: AUD2EN: 1 = Enable audio channel 2 DMA (start sound); 0 = Disable audio channel 2 DMA (stop sound)

Bit 3: AUD3EN: 1 = Enable audio channel 3 DMA (start sound); 0 = Disable audio channel 3 DMA (stop sound)

Bit 4: DSKEN: 1 = Enable disk DMA; 0 = Disable disk DMA

Bit 5: SPREN: 1 = Enable sprite DMA; 0 = Disable sprite DMA

Bit 6: BLTEN: 1 = Enable blitter DMA; 0 = Disable blitter DMA

Bit 7: COPEN: 1 = Enable copper DMA; Disable copper DMA

Bit 8: BPLEN: 1 = Enable bit-plane DMA; 0 = Disable bit-plane DMA

Bit 9: DMAEN: Master DMA off switch (0 = Disables DMA for all channels listed above)

Bit 10: BLTPRI: Blitter priority (1 = Gives blitter full priority over the 68000; 0 = Gives blitter partial priority over the 68000)

Bit 11: Unassigned

Bit 12: Unassigned

Bit 13: BZERO: Has no effect when written to. See DMACONR ($DFF002)
Bit 14: BBUSY: Has no effect when written to. See DMACONR ($DFF002)
Bit 15: SET/CLR: Set or clear bits of this register (1 = Bits written with 1 will be set; 0 = Bits written with a 1 will be cleared)

This register turns on and off DMA, *Direct Memory Access*. DMA is the process by which the custom chips access memory without the use of the computer's microprocessor.

DMA relieves the Amiga's 68000 microprocessor of several memory-intensive tasks, such as audio production, graphics manipulation, and disk read/write functions. There are six DMA channels in all. One channel is reserved for each of the following: audio, disk, sprite, bit-plane, copper, and blitter operations. This register controls all six DMA channels.

What happens when you write to this register depends on the way you set bit 15. If it is set to 0, any other bit that is written to with a 1 will be cleared (equal to 0). If bit 15 is set to 1, any other bit that is written to with a 1 will be set (equal to 1). Setting bits 0-8 of this register enables the corresponding DMA channel. For example, to activate the DMA for audio channel 0, store a $8001 into this register. Use $0001 to turn it off.

To check which DMA channels are enabled and which are disabled, use DMACON's companion register, DMACONR ($DFF002).

Bits 0-3. These lower four bits control the DMA for each of the Amiga's four audio channels. Enabling an audio channel's DMA initiates the output of the channel's waveform. In other words, the Amiga starts making noise. To turn a sound off, simply disable the channel's DMA. For details, refer to locations $DFF09E-$DFF0DE.

Bit 4. This bit controls disk DMA, which handles the transfer of data between the floppy disk drive and RAM. This bit is used in conjunction with the DMAEN bit in the disk controller's DSKLEN register ($DFF024). Both bits must be set in order for disk DMA to become active. To avoid destroying data, be sure that any previous disk operation is complete before you activate disk DMA.

Bit 5. Sprite DMA must be enabled in order for any sprites to be displayed. The library function macros ON_SPRITE and OFF_SPRITE may be used to set and clear this bit, respectively. If you're programming by the book (that is, you haven't thrown out the operating system), sprite DMA should already be on; the operating system leaves the sprites on all the time because the mouse pointer is a sprite.

Bit 6. This is the blitter's on-off switch. If this bit is set, the blitter can begin copying data. Clear this bit and you disable the blitter. During normal operation, this bit stays set while the activation of blitter activity is controlled via the BLTSIZE register ($DFF058).

Bit 7. This bit enables or disables the copper's ability to execute instructions. You can produce some very interesting results if you clear this bit (copper DMA off). Without the aid of the copper, the Amiga's screen display becomes quite confused.

Bit 8. When this bit is set equal to 1 (and it almost always is), bitplane DMA reads data from the chip memory specified by registers BPL1PTH- BPL6PTL ($DFF0E0-$DFF0F6) into registers BPL1DAT-BPL6DAT ($DFF0110- $DFF11A), where their values are used by the video hardware to construct the physical screen display. Turn this bit off and you turn off the screen display, sprites and all. The entire screen takes on the background color, which may change according to the current copper list. In fact, a simple way to blank the screen is to set register COLOR00 ($DFF180) to zero (black) and then turn off both copper and bitplane DMA by storing a $0180 into this register. Use the value $8180 to turn the DMA back on.

Bit 9. This bit controls the DMA for all six channels. If this bit is clear, all DMA is disabled, regardless of the value of bits 0-8. For all practical purposes, you should never clear this bit. If you do, the screen blanks, sound production stops, and floppy disk operations go berserk. In a sense, disabling DMA disables the Amiga.

Bit 10. Setting this bit (which Commodore's *Hardware Reference Manual* affectionately refers to as the "blitter-nasty" bit) gives the blitter complete priority over the 68000. If this bit is set during heavy blitter operation, the 68000 will not be able to access chip RAM or any of the custom hardware registers. Normally this bit is clear, giving the microprocessor access to chip RAM and the custom hardware for at least one out of every four bus cycles. If you're performing time-critical blitter operations and you don't care about what the microprocessor is up to, this bit can be very useful.

$DFF098 CLXCON

Collision Control

Status: Write-Only. Chip: Denise

Bit 0: MVP1: Match value for bitplane 1 collisions
Bit 1: MVP2: Match value for bitplane 2 collisions
Bit 2: MVP3: Match value for bitplane 3 collisions
Bit 3: MVP4: Match value for bitplane 4 collisions
Bit 4: MVP5: Match value for bitplane 5 collisions
Bit 5: MVP6: Match value for bitplane 6 collisions
Bit 6: ENBP1: 1 = Enable collision detection for bitplane 1
Bit 7: ENBP2: 1 = Enable collision detection for bitplane 2
Bit 8: ENBP3: 1 = Enable collision detection for bitplane 3
Bit 9: ENBP4: 1 = Enable collision detection for bitplane 4

Bit 10: ENBP5: 1 = Enable collision detection for bitplane 5
Bit 11: ENBP6: 1 = Enable collision detection for bitplane 6
Bit 12: ENSP1: 1 = Enable collision detection for sprite 1
Bit 13: ENSP3: 1 = Enable collision detection for sprite 3
Bit 14: ENSP5: 1 = Enable collision detection for sprite 5
Bit 15: ENSP7: 1 = Enable collision detection for sprite 7

The Amiga can detect three types of graphic collisions: sprite to sprite, sprite to bitplane, and bitplane to bit plane. The CLXDAT register ($DFF00E) tells you when and what type of a collision is taking place. What constitutes a collision depends on how you set this register.

A *collision* can be defined as anything from the contact between a sprite and a pixel of a specific color to the contact between a sprite or playfield and anything (an impractical yet interesting situation).

Bits 0-5. These bits specify exactly what value—0 or 1—a bit-plane's pixel must be in order for an object to be able to collide with it. This match value can be different for each of the six possible bit planes. A bit plane's match value is taken into account only if that bitplane's corresponding collision-enable bit is set (see bits 6-11 below).

Bits 6-11. These bits determine which bit planes are used in collision detection. If a bit plane's collision-enable bit is 0, that bit plane is ignored; its bits have no say as to whether a collision takes place or not. Oddly, however, the Amiga always seems to think that a bitplane collision is taking place when all of these bits are set to zero.

Through creative use of bits 0-5 and bits 6-11 you can specify that collisions only take place with pixels of a certain color or within a certain range of colors. As a simple example, let's say you open a one-bitplane screen and that you enable collision-detection for that bit plane (bitplane 1). Your background color (for 0 pixels) is black. Your foreground color (for 1 pixels) is white. If this register's 0 bit is set to 1, bit-plane collisions only occur with white foreground pixels. Conversely, if bit 0 is set to 0, bit-plane collisions only occur with black background pixels.

Obviously, multiple-bitplane screens complicate matters. Not only must the pixels for bitplane 1 be checked for the correct value, so must the pixels for all the other bit planes with collision detection enabled. The following table shows which pixels may cause collisions under a variety of conditions using a six-bitplane screen:

Match Value	Enabled Bitplanes	Pixel Values That Cause Collisions
xxxxxx	000000	All pixels can cause a collision
1111xx	111100	111100, 111101, 111110, 111111
01xx10	110011	010010, 010110, 011010, 011110
xx1111	001111	001111, 011111, 101111, 111111
000000	111111	000000
101010	111111	101010
111111	111111	111111

x = doesn't matter

For accurate collision results, never enable collision detection for bitplanes that are not in use.

Bits 12-15. Setting these bits allows you to check for collisions involving odd-numbered sprites. (Collision detection is always enabled for even-numbered sprites.) If you set bit 12, for example, you can use the Amiga's CLXDAT register to detect if sprite 1 collides with something. Actually, you can only detect if sprite 1 *or* sprite 0 collides with something, because the CLXDAT register assigns just one bit to every two sprites. See hardware location $DFF00E for details.

$DFF09A INTENA

Interrupt Enable

Status: Write-Only. Chip: Paula

Bit 0: TBE: 1 = Enable serial transmit buffer empty interrupt (level 1)
Bit 1: DSKBLK: 1 = Enable disk block done interrupt (level 1)
Bit 2: SOFT: 1 = Enable software interrupts (level 1)
Bit 3: PORTS: 1 = Enable CIA-A and expansion port interrupts (level 2)
Bit 4: COPER: 1 = Enable copper interrupt (level 3)
Bit 5: VERTB: 1 = Enable vertical blank interrupt (level 3)
Bit 6: BLIT: 1 = Enable blitter ready interrupt (level 3)
Bit 7: AUD0: 1 = Enable audio channel 0 interrupt (level 4)
Bit 8: AUD1: 1 = Enable audio channel 1 interrupt (level 4)
Bit 9: AUD2: 1 = Enable audio channel 2 interrupt (level 4)
Bit 10: AUD3: 1 = Enable audio channel 3 interrupt (level 4)
Bit 11: RBF: 1 = Enable serial receive buffer full interrupt (level 5)
Bit 12: DSKSYN: 1 = Enable disk sync pattern found interrupt (level 5)
Bit 13: EXTER: 1 = Enable CIA-B and expansion port interrupts (level 6)
Bit 14: INTEN: Master interrupt off switch (0 = Disables all interrupts listed above)
Bit 15: SET/CLR: Set or clear bits of this register (1 = Bits written with 1 will be set; 0 = Bits written with a 1 will be cleared)

What happens when you write to this register depends on the way you set bit 15. If it is set to 0, any other bit that is written to with a 1 will be cleared (equal to 0). If bit 15 is set to 1, any other bit that is written to with a 1 will be set (equal to 1). Setting bits 0-13 of this register enables the corresponding interrupt (allows it to take place). For example, to enable software interrupts, store a $8004 into this register. Use $0004 to disable software interrupts.

To check which interrupts are enabled and which are disabled, use register INTENAR ($DFF01C). To force an interrupt to take place, use register INTREQ ($DFF09C). To find out which type of interrupt is or is not taking place, use register INTREQR ($DFF01E).

Bits 0-13. See register INTENAR ($DFF01C) for details.

Bit 14. If this bit is clear, all maskable interrupts are disabled, regardless of the value of bits 0-13. For all practical purposes, you should never alter this bit directly. The Amiga's operating system often relies on the value of this bit being 0. If you set this bit to 1 when the operating system expects all maskable interrupts to be disabled, the Amiga will most certainly crash. Therefore, unless you have thrown out the operating system and are handling interrupts yourself, use the exec.library functions Disable( ) and Enable( ) to control the status of this bit.

$DFF09C INTREQ

Interrupt Request
Status: Write-Only. Chip: Paula

Bit 0: TBE: 1 = Request a serial transmit buffer empty interrupt (level 1);
0 = Clear this interrupt

Bit 1: DSKBLK: 1 = Request a disk block done interrupt (level 1);
0 = Clear this interrupt

Bit 2: SOFT: 1 = Request a software interrupts (level 1);
0 = Clear this interrupt

Bit 3: PORTS: 1 = Request a CIA-A or an expansion port interrupt (level 2);
0 = Clear this interrupt

Bit 4: COPER: 1 = Request a copper interrupt (level 3);
0 = Clear this interrupt

Bit 5: VERTB: 1 = Request a vertical blank interrupt (level 3);
0 = Clear this interrupt

Bit 6: BLIT: 1 = Request a blitter ready interrupt (level 3);
0 = Clear this interrupt

Bit 7: AUD0: 1 = Request an audio channel 0 interrupt (level 4);
0 = Clear this interrupt

Bit 8: AUD1: 1 = Request an audio channel 1 interrupt (level 4);
0 = Clear this interrupt
Bit 9: AUD2: 1 = Request an audio channel 2 interrupt (level 4);
0 = Clear this interrupt
Bit 10: AUD3: 1 = Request an audio channel 3 interrupt (level 4);
0 = Clear this interrupt
Bit 11: RBF: 1 = Request a serial receive buffer full interrupt (level 5);
0 = Clear this interrupt
Bit 12: DSKSYN: 1 = Request a disk sync pattern found interrupt (level 5);
0 = Clear this interrupt
Bit 13: EXTER: 1 = Request a CIA-B or an expansion port interrupt (level 6);
0 = Clear this interrupt
Bit 14: INTEN: 1 = Request a level 6 interrupt;
0 = Clear this interrupt
Bit 15: SET/CLR: Set or clear bits of this register
1 = Bits written with 1 will be set;
0 = Bits written with a 1 will be cleared)

If you prefer, you can force the microprocessor to generate an interrupt even though the actual interrupt conditions have not been met. To do this, simply set the bit that corresponds with the interrupt you wish to trigger. This is exactly how the copper generates its level 3 interrupt—by storing a $8010 into this register using a copperlist MOVE command. (In fact, you can use copper lists to generate any of these interrupts.)

For the generation of software interrupts, the exec.library offers the Cause( ) function. Among other things, this function sets bit 2 in this register.

Requesting an interrupt may not be enough, however. The corresponding bit in the INTENA register ($DFF09A) must also be set. Only when a bit in both registers is set, *and* INTENA's bit 14 is set, will an interrupt take place.

After an interrupt has been processed, you must clear that interrupt so that others may take place. You clear an interrupt by setting its bit to 0.

What happens when you write to this register depends on the way you set bit 15. If it is set to 0, any other bit that is written to with a 1 will be cleared (equal to 0). If bit 15 is set to 1, any other bit that is written to with a 1 will be set (equal to 1). Setting bits 0-13 of this register requests the corresponding interrupt. For example, to request a transmit buffer empty interrupt, store a $8001 into this register. Use $0001 to clear it.

To enable or disable an interrupt, use register INTENA ($DFF09A). To check which interrupts are enabled and which are disabled, use register INTENAR ($DFF01C). To find out which type of interrupt is or is not taking place, use register INTREQR ($DFF01E).

Location Range: $DFF09E-$DFF0DE

Audio Registers

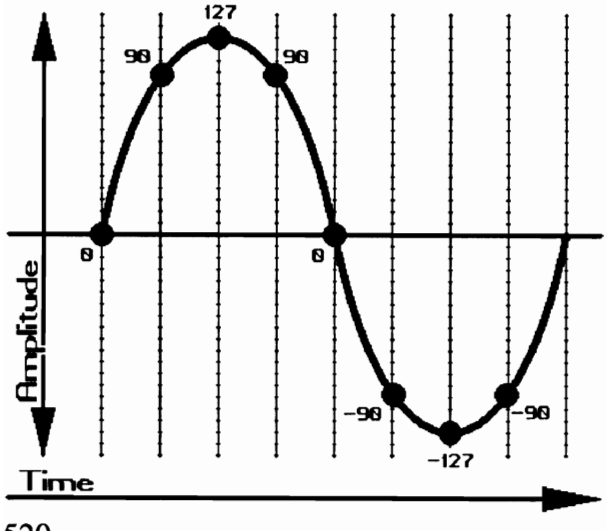
Hardware locations $DFF09E-$DFF0DE contain the Amiga's sound registers.

The Amiga has four separate sound channels numbered 0-3. Channels 0 and 3 are sent through the Amiga's left audio output port; channels 1 and 2 are sent through the Amiga's right audio output port. This allows the computer to simultaneously play up to two digitized sounds through each of its stereo outputs.

Amiga sound relies on *waveform sampling*—a method in which a sound's waveform is described digitally in memory. On the Amiga, waveforms are represented by a series of bytes stored in the computer's chip RAM. These bytes may be obtained from a specialized analog-to-digital (A-to-D) converter (more commonly known as a sound sampler or sound digitizer), mathematically calculated via trigonometric functions, derived from a preinitialized table of data, or simply made up.

Figure 3-4 shows how a simple sine wave might be represented. Points on the waveform are described by a value ranging between -128 and +127 (if the byte's high bit is set, the lower 7 bits are interpreted as a negative value). Each byte reflects the waveform's amplitude (height) at a particular moment in time. The accuracy of a waveform sample depends upon its *sampling rate* (how often the waveform's *amplitude* is measured). The higher the sampling rate, the more accurate and better-sounding the waveform, and the more memory needed to store the waveform. The Amiga can play sounds with sampling rates as high as 28 kilohertz—approximately 28,000 samples per second.

Figure 3-4: Sine Wave Sample



Once you have a waveform sample loaded into chip RAM, turning that data into sound is relatively simple. Just tell the computer where your waveform is located and the length of the sample in words (number of bytes divided by 2). Next, tell the computer how frequently to output the sampled data (how fast to play the sound), set the volume, and then signal that the sound should begin by turning on DMA for the desired audio channel (see register DMACON, $DFF096). The Amiga's sound hardware now takes over. In fact, it will continue to play your sound—repeating it if necessary—until you tell it to stop.

The Amiga has a few tricks up its sleeve when it comes to modifying sampled waveforms. By linking two audio channels together, you can modulate a sound's volume and/or period (frequency). This gives you the ability to create truly customizable sound envelopes and finely detailed tremolo and pitch-bend effects. And because of the way the audio DMA operates, it's possible to combine two separate waveforms into one continuous sound, even when the waveforms are of different length or are located in noncontiguous memory.

Although direct manipulation of the audio hardware is the most effective way to produce sound on the Amiga, it's not recommended by Commodore. Because of the multitasking nature of the computer, grabbing audio channels without permission from the operating system is unfriendly, to say the least. For explanatory purposes, the audio programs presented here access the hardware directly. If you want your software to run concurrently with other programs that produce sound, you should use the functions provided by the Amiga's audio device.

All of the Amiga's audio registers are write-only. That means that their values cannot be determined by reading these locations. It should also be noted that, besides controlling audio modulation, register ADKCON ($DFF09E) affects the operation of the Amiga's floppy disk controller and the break status of the serial port.

$DFF09E ADKCON

Audio/Disk Control

Status: Write-Only. Chip: Paula

Bit 0:	ATVOL0: 1 = Audio channel 0 modulates the volume of channel 1
Bit 1:	ATVOL1: 1 = Audio channel 1 modulates the volume of channel 2
Bit 2:	ATVOL2: 1 = Audio channel 2 modulates the volume of channel 3
Bit 3:	ATVOL3: 1 = Disables the output of audio channel 3
Bit 4:	ATPER0: 1 = Audio channel 0 modulates the period of channel 1
Bit 5:	ATPER1: 1 = Audio channel 1 modulates the period of channel 2
Bit 6:	ATPER2: 1 = Audio channel 2 modulates the period of channel 3

Bit 7:	ATPER3: 1 = Disables the output of audio channel 3
Bit 8:	FAST: 1 = MFM, two-microseconds-per-bit disk operation; 0 = GCR, four-microseconds-per-bit disk operation (not used by AmigaDOS)
Bit 9:	MSBSYNC: 1 = Enables GCR-format synchronization for disk operations (not used by AmigaDOS)
Bit 10:	WORDSYNC: 1 = Forces disk controller to synchronize on the data word written to the DSKSYNC register ($DFF07E)
Bit 11:	UARTBRK: 1 = Sends an RS-232 break-signal by setting the serial port's TXD line to 0
Bit 12:	MFMPREC: 1 = Selects MFM disk format; 0 = Selects GCR disk format (not used by AmigaDOS)
Bits 13-14:	PRECOMP: Disk precompensation time (00 = None; 01 = 140 nanoseconds; 10 = 280 nanoseconds; 11 = 560 nanoseconds)
Bit 15:	SET/CLR: Set or clear bits of this register (1 = Bits written with 1 will be set; 0 = Bits written with a 1 will be cleared)

What happens when you write to this register depends on the way you set bit 15. If it is set to 0, any other bit that is written to with a 1 will be cleared (equal to 0). If bit 15 is set to 1, any other bit that is written to with a 1 will be set (equal to 1).

Bits 0-7. The first byte of this word-long register controls the modulation status of all four audio channels. The Amiga can use the output from one channel to *modulate* (alter) the sound of the next higher channel—channel 0 can modulate the output of channel 1, channel 1 can modulate the output of channel 2, and channel 2 can modulate the output of channel 3.

A modulating channel makes no sound of its own; instead, its waveform data is interpreted as the volume and/or period values for the channel above it. For example, let's say that channel 0's waveform data gradually increases and decreases in value. If you set channel 0's volume modulation bit, the sound produced by channel 1 will waver up and down in loudness. Turn on period modulation, and channel 1's sound will waver in pitch, creating a tremolo effect. If you like, you can modulate a channel's volume and period at the same time.

Using a channel as a modulator is very similar to using it as a sound generator except that its output isn't heard—it's used to modulate the next channel. As usual, you must set the channel's period to determine how frequently the data should be sampled,

enable DMA to begin output, and then turn off DMA when you're done. You must also tell the computer where the channel's data is stored. But the data is interpreted differently. Normal waveform data is treated as a series of bytes. When you use a channel as a volume modulator, its data is treated as a series of 16-bit words, but only the low 7 bits of each word is used:

Volume Modulation Data

Bits	Purpose
0-6	Volume data (0-64)
7-15	Ignored

When you use a channel as a period modulator, its data is treated as a series of 16-bit words, every bit of which is significant:

Period Modulation Data

Bits	Purpose
0-15	Period data

When a channel is used to modulate both volume and period, the data words alternate between volume and period data. The first word represents volume data, the second word represents period data, the third word represents volume data, the fourth word represents period data, the fifth word represents volume data. . . and so on and so forth.

Because there is no audio channel higher than channel 3, setting either of this channel's modulation bits simply disables channel 3's output.

Bit 8. The Amiga disk controller supports two formats for storing data on a disk: MFM and GCR. AmigaDOS uses MFM format, which fits less data on a disk than GCR, but operates at twice the speed (two microseconds per bit vs. four microseconds per bit). Setting bit 8 reduces the controller's speed for GCR operations.

Bit 9. Setting this bit to 1 informs the disk controller to synchronize itself using GCR conventions. (Synchronizing enables the controller to accurately align itself with the data stored on the disk.) Specifically, it looks for nine or more consecutive 1 bits as its sync pattern. Note that AmigaDOS does not support the GCR format.

Bit 10. When this bit is set to 1, the disk controller's DMA is enabled and the controller prepares to search the disk for the sync pattern—a 16-bit word that signifies the beginning of data—written to the DSKSYNC register ($DFF07E). The disk controller does not start searching until DSKSYNC is written to. When the pattern is

found, subsequent data is read into RAM. Bit 12 of the DSKBYTR register ($DFF01A) is set to 1 for two or four microseconds (depending on the setting of ADKCON's bit 8) as soon as the sync pattern is located. This event can also be used to trigger a level 6 interrupt.

In MFM format, the value of the disk's sync pattern depends on what the programmer or operating system stores in the DSKSYNC register ($DFF07E). By convention, this 16-bit value should be one that is impossible to create using MFM data coding. This way, it can't be confused with actual data. AmigaDOS uses the number $4489 as its sync pattern.

Bit 11. Setting this bit to 1 interrupts the Amiga's serial output and sets the serial port's TXD line to 0.

Bit 12. This bit allows you to choose between MFM and GCR disk formats. If this bit is cleared, GCR format is used. If this bit is set, MFM format is used. AmigaDOS uses MFM encoding. Briefly put, MFM format inserts clock bits between each data bit written to disk. If two consecutive data bits are zero, a 1 clock bit is put between them. If either of two consecutive data bits is 1, a 0 clock bit is put between them. Using this coding method, the amount of data written to a disk is doubled, but the accuracy of reading that data is greatly increased. You see, the disk controller has an easier time synchronizing on data that changes often (bits that change from 0 to 1 or 1 to 0) than on data that repeats (say, a long series of 1 bits or a long series of 0 bits).

GCR format handles synchronization problems by encoding every four bits of data into five bits. The result is a binary number with no more than two adjacent 0 bits and no more than eight adjacent 1 bits. The following table shows the five-bit GCR equivalents of the binary numbers 0000 through 1111.

Binary Data	GCR Equivalent	Binary Data	GCR Equivalent
0000	01010	1000	01001
0001	01011	1001	11001
0010	10010	1010	11010
0011	10011	1011	11011
0100	01110	1100	01101
0101	01111	1101	11101
0110	10110	1110	11110
0111	10111	1111	10101

The disk controller has more trouble synchronizing on GCR encoded data than on MFM encoded data. As a result, the controller's speed should be decreased to 4 microseconds per bit when operating in GCR mode (see bit 8).

Bit 13-14. These bits set the disk controller's precompensation time, which directly affects the density of data written to disk. The faster the precompensation time, the more closer together the bits are placed. With AmigaDOS, the data density is highest on the inside tracks, probably due to the fact that there is less physical space for these tracks.

$DFF0A0 AUD0LC ($DFF0A0 = AUD0LCH and $DFF0A2 = AUD0LCL)

Channel 0 Waveform Address
Status: Write-Only. Chip: Agnus

This is where you must store the starting address of your waveform data for audio channel 0 prior to turning on the channel's DMA. AUD0LC actually consists two separate hardware registers—AUD0LCH (H for the High bits of the waveform's address) and AUD0LCL (L for the Low bits of the waveform's address). Since they're mapped as two consecutive memory locations, it's easiest if your program simply treats them as one 32-bit register. The low bit of this register is always interpreted as 0, so audio data must begin at an even memory location.

$DFF0A4 AUD0LEN

Channel 0 Waveform Length
Status: Write-Only. Chip: Paula

The length of your waveform data in words (number of bytes divided by 2) goes into this register (the audio DMA grabs waveform data two bytes at a time). The fact that this 16-bit register accepts the waveform length in words, not bytes, implies two things: First, your waveform should be an even number of bytes in length. Second, your waveform data may not exceed 131,070 (65,535 times 2) bytes.

However, it is possible to play waveforms longer than 131,070 bytes, thanks the way the audio hardware works. The Amiga buffers the contents of the AUDxLC and AUDxLEN registers (where *x* denotes the desired audio channel, 0-3), using backup registers. The computer copies the main registers into the backup registers every time it begins to play a waveform. This allows you to change the AUDxLC and AUDxLEN registers without affecting the current sound. If you change these registers before the current waveform repeats, the computer will continue playing using the new waveform. Using this technique, you can play extremely long sound samples. The number of waveforms that you play in succession like this is limited only by memory.

Changing the AUDxLC and AUDxLEN registers on the fly requires accurate timing. You must be sure that audio DMA has read the register's current settings before

you change their value. The easiest way to do this is to simply wait until you think it's safe. You can base your delay loop on your sound's period (AUDxPER), pausing only long enough for the first few bytes of waveform data to be output. Your other option involves the use of interrupts.

Any of the Amiga's audio channels can cause a level 4 interrupt just before your sound starts a new waveform cycle. When this interrupt occurs, you know it's safe to change the AUDxLC and AUDxLEN registers. Be warned: Interrupts caused by high-frequency waveforms can occur very often—possibly too often for the Amiga's operating system to keep up. It's best to leave audio interrupts alone, unless you absolutely need them. See the INTENA and INTREQ registers ($DFF09A and $DFF09C) for more information on interrupts.

$DFF0A6 AUDOPER

Channel 0 Period

Status: Write-Only. Chip: Paula

A sound's *frequency* (how fast it's played) is determined by the value stored in this register. The lower the value, the higher the frequency. The lowest value you should ever use is 124, since this is about as fast as the Amiga can go. If you use a value less than 124, you run the risk of having the same two bytes of data output twice in a row. Since this is a 16-bit register, 65,535 is the largest period value possible.

The computer uses the period value as a counter that it decrements every .279365 microseconds. Every time the period counter reaches 0, the Amiga outputs a byte of waveform data, the period counter is reset to its initial value, the value of the length register (AUDxLEN) is decremented by one, and the countdown resumes. When the length register hits zero, the whole process starts over—the AUDxLC and AUDxLEN registers are reloaded, as is the period counter—and the sound repeats. This process continues until you stop the sound by disabling the audio channel's DMA via the ADKCON register.

How you determine a sound's period value depends on how you get your waveform data. If you're like most people, you use a digitizer to create most of your sounds. If this is the case, you'll want to output your waveform data at the same rate it was digitized.

Most sound-digitizing software specifies waveform frequency by the number of samples per second, not by the period value that the Amiga's hardware expects. The equation for converting samples-per-second to period value is

Period = 3579546 / Samples-Per-Second

So, if you want to play a sound that was sampled at 12 kHz (12,000 samples per second), use a period value of 298 (3579546 / 12,000).

Why does this equation work? If the Amiga decrements its period register every .279365 microseconds, 3,579,546 (1 divided by .000000279365) decrements take place every second. So, to waste enough time between samples in a 12,000 samples-per-second sound, the computer must pause for 3579564 / 12000 decrements between every byte of waveform data.

If your waveform data represents one complete cycle of a waveform, such as the sine wave shown in Figure 3-4, you'll probably want to adjust your period rate to play various musical notes.

To calculate the period value for a particular note, you must take two things into account: the size of the waveform in bytes and the frequency value of the desired note. The following table gives the approximate frequency value for all 12 notes in a chromatic scale played in the fourth octave (middle C is a C in the fourth octave).

Note	Frequency
A	440
A# or Bb	466
B	494
C	523
C# or Db	554
D	587
D# or Eb	622
E	659
F	699
F# or Gb	740
G	784
G# or Ab	831

To calculate the period value for a note, use the equation

Period = 3579546 / (Length * Frequency)

where **Length** is the size of the waveform in bytes and **Frequency** is the note's frequency value obtained from the table above. For example, to play a C note using the sine wave shown in Figure 3-4, you would use a period value of 855, because the integer result of 379546 / (8 * 523) is 855.

To obtain notes in the next higher octave (octave 5), divide your period value by 2. Every time you divide the period by 2, you raise the note by an octave. Conversely, every time you multiply the period value by 2, you bring the note down by an octave. Using the sine wave example above, a period value of 3420 (855 * 2 * 2) plays a very low C in the second octave.

Please remember: For this note-to-period equation to be accurate, your sound data must describe one complete cycle of a waveform. This equation works best on relatively simple waveforms—sine, triangle, sawtooth, etc.

If you don't want your sound to repeat, you should consider one more thing when calculating a period value: when to stop the sound. You see, the faster you tell the Amiga to play a sound, the faster it's going to finish. To figure out just how long you should let your sound play, use the following equation:

Microseconds = .279365 * Period * Length

where **Period** is the period value used to play the sound and **Length** is the length of your waveform data in bytes (not words). **Microseconds**, as you might have guessed, is the number of microseconds it takes to play your sound once.

$DFF0A8 AUDOVOL

Channel 0 Volume

Status: Write-Only. Chip: Paula

This volume register can accept values between 0 and 64, where 0 is the lowest volume setting (inaudible, in fact) and 64 is the highest volume setting. A sound's volume also depends on the waveform's amplitude (height), so two sounds with equal volume settings may not always play with equal loudness.

The following table shows the affect in decibels that the volume has on a sound:

Volume	Decibels	Volume	Decibels	Volume	Decibels	Volume	Decibels
0	-infinity	17	-11.5	34	-5.5	51	-2.0
1	-36.1	18	-11.0	35	-5.2	52	-1.8
2	-30.1	19	-10.5	36	-5.0	53	-1.6
3	-26.6	20	-10.1	37	-4.8	54	-1.5
4	-24.1	21	-9.7	38	-4.5	55	-1.3
5	-22.1	22	-9.3	39	-4.3	56	-1.2
6	-20.6	23	-8.9	40	-4.1	57	-1.0
7	-19.2	24	-8.5	41	-3.9	58	-0.9
8	-18.1	25	-8.2	42	-3.7	59	-0.7
9	-17.0	26	-7.8	43	-3.5	60	-0.6
10	-16.1	27	-7.5	44	-3.3	61	-0.4
11	-15.3	28	-7.2	45	-3.1	62	-0.3
12	-14.5	29	-6.9	46	-2.9	63	-0.1
13	-13.8	30	-6.6	47	-2.7	64	-0.0
14	-13.2	31	-6.3	48	-2.5		
15	-12.8	32	-6.0	49	-2.3		
16	-12.0	33	-5.8	50	-2.1		

$DFF0AA AUD0DAT

Channel 0 Data

Status: Write-Only. Chip: Paula

This is the data buffer used for audio channel 0. Ordinarily, you won't ever have to use this register; it's used by the Amiga to automatically output waveform data while a sound is in progress. However, if you want to experiment with non-DMA sound creation, this register allows you to manually output waveform data to the Amiga's D-to-A converter. You cannot write to this register unless the channel's audio DMA is off.

To generate sounds manually, you must store two bytes worth of waveform data into this register at a rate fast enough to keep up with the audio channel's period rate. (The register's high byte is output first; the low byte follows.) If you output data too slowly, the audio channel continues to play the last byte sent to it.

To help you keep in sync, level 4 interrupts occur every time this register is ready to accept more data. (Normally, when audio DMA is on, level 4 interrupts occur only when the *entire* waveform has been output.) Because sound frequencies are usually very high, audio interrupts that are used to manually output waveform data occur at an extremely frequent rate.

Besides being cumbersome, producing sound manually via data registers eats up a considerable amount of processor time. It's highly recommended that you take advantage of the Amiga's audio DMA instead.

Location Range: $DFF0AC-$DFF0AE

Unused

The Amiga reserves enough space for eight word-long audio registers for each channel, but it only needs six. The remaining addresses are unused.

Location range: $DFF0B0-$DFF0DE

Audio Channel 1-3

The operation of these channels' registers are identical to those of audio channel 0.

$DFF0B0 AUD1LC ($DFF0B0 = AUD1LCH and $DFF0B2 = AUD1LCL)

Channel 1 Waveform Address
Status: Write-Only. Chip: Agnus

$DFF0B4 AUD1LEN

Channel 1 Waveform Length
Status: Write-Only. Chip: Paula

$DFF0B6 AUD1PER

Channel 1 Period
Status: Write-Only. Chip: Paula

$DFF0B8 AUD1VOL

Channel 1 Volume
Status: Write-Only. Chip: Paula

$DFF0BA AUD1DAT

Channel 1 Data
Status: Write-Only. Chip: Paula

Location range: $DFF0BC-$DFF0BE

Unused

$DFF0C0 AUD2LC ($DFF0C0 = AUD2LCH and $DFF0C2 = AUD2LCL)

Channel 2 Waveform Address
Status: Write-Only. Chip: Agnus

$DFF0C4 AUD2LEN
Channel 2 Waveform Length
Status: Write-Only. Chip: Paula

$DFF0C6 AUD2PER
Channel 2 Period
Status: Write-Only. Chip: Paula

$DFF0C8 AUD2VOL
Channel 2 Volume
Status: Write-Only. Chip: Paula

$DFF0CA AUD2DAT
Channel 2 Data
Status: Write-Only. Chip: Paula

Location range: $DFF0CC-$DFF0CE

Unused

$DFF0D0 AUD3LC ($DFF0D0 = AUD3LCH and $DFF0D2 = AUD3LCL)
Channel 3 Waveform Address
Status: Write-Only. Chip: Agnus

$DFF0D4 AUD3LEN
Channel 3 Waveform Length
Status: Write-Only. Chip: Paula

$DFF0D6 AUD3PER
Channel 3 Period
Status: Write-Only. Chip: Paula

$DFF0D8 AUD3VOL
Channel 3 Volume
Status: Write-Only. Chip: Paula

$DFF0DA AUD3DAT
Channel 3 Data
Status: Write-Only. Chip: Paula

Location range: $DFF0DC-$DFF0DE
Unused

Location range: $DFF0E0-$DFF0F6
Bitplane Pointer Registers
These registers point to the current word that is being displayed from each bitplane. They are set by the programmer, but they increment as each word is fetched by the bitplane DMA system. Therefore, these registers must be reset for each screen. This is usually done in the copper list, but it may also be written to directly by the programmer during the vertical blank interrupt. The initial value store here should be the starting address of the bitplane.

These registers may be thought of as either a series of six long word registers or as a series of six pairs of word registers. Since the copper can only write word values, it will take two copper instructions to set a single bitplane pointer. The lowest bit in a bitplane pointer is ignored, so bitplanes must start on word boundaries. Of course, bitplanes must also be stored in chip RAM.

$DFF0E0 BPL1PT ($DFF0E0 = BPL1PTH and $DFF0E2 = BPL1PTL)
Bitplane 1 Pointer
Status: Write-Only. Chip: Agnus

$DFF0E4 BPL2PT ($DFF0E4 = BPL2PTH and $DFF0E6 = BPL2PTL)
Bitplane 2 Pointer
Status: Write-Only. Chip: Agnus

$DFF0E8 BPL3PT ($DFF0E8 = BPL3PTH and $DFF0EA = BPL3PTL)
Bitplane 3 Pointer
Status: Write-Only. Chip: Agnus

$DFF0EC BPL4PT ($DFF0EC = BPL4PTH and $DFF0EE = BPL4PTL)
Bitplane 4 Pointer
Status: Write-Only. Chip: Agnus

$DFF0F0 BPL5PT ($DFF0F0 = BPL5PTH and $DFF0F2 = BPL5PTL)

Bitplane 5 Pointer
Status: Write-Only. Chip: Agnus

$DFF0F4 BPL6PT ($DFF0F4 = BPL6PTH and $DFF0F6 = BPL6PTL)

Bitplane 6 Pointer
Status: Write-Only. Chip: Agnus

Location range: $DFF0F8-$DFF0FE

Unused

Location range: $DFF100-$DFF104

Bitplane Control Registers

These three registers control the modes of the Amiga video display. Within the bits of these registers are the means of creating just about any Amiga screen possible. BPLCON0 ($DFF100) controls how the pixels are displayed. Genlocking, interlace, dual-playfield mode, HAM, Extra-HalfBrite, and Hi-Res are all among its domain. BPLCON1 ($DFF102) controls the hardware smooth-scrolling of the Amiga. BPLCON2 ($DFF104) is the priority register which determines how the playfield and sprites will interact.

We'll need to define a few terms in order to be able to discuss these registers.

Dual-playfield mode. In Dual-playfield mode, the odd-numbered bitplanes are grouped into one playfield (playfield 1) and the even-numbered bitplanes are grouped into another playfield (playfield 2). One playfield has priority over the other, which means that the backgroup playfield shows through only when the foreground playfield's pixel is color 0.

Extra-HalfBrite (EHB). In EHB, pixels may have any value in the range 0-63. Pixels in the range 0-31 use the 32 color registers normally. Pixels in the range 32-63 use the lower 32 color registers, but with the red, green, and blue values each shifted one bit to the right (divided by 2).

Genlock. This is a method of combining computer graphics with non-computer video, such as that obtained from a VCR, television tuner, or video camera.

Hold And Modify (HAM). In HAM mode, 16 color registers (4 bitplanes) are used normally. In 5-bitplane HAM mode, a bit set in the fifth bitplane signals that the pixel should use the color of the pixel immediately to the left, but with the blue value corresponding to the value of the lower four bitplanes. In 6-bitplane mode, the bits in the fifth and sixth bitplanes are used to specify whether the lower sixteen color registers

should be used normally, or whether the pixel should be based on the pixel immediately to the left, with the red, blue, or green intensity altered. HAM images are the most photographic available on the Amiga. Typically, six-bitplane HAM is used, with five-bitplane HAM being a rarity.

Interlace. This is a method of doubling the vertical resolution of a screen by showing alternating lines on alternating scans. Odd lines are displayed in one pass, even lines on the next. For details on how interlace screens work, see the VPOSR entry at $DFF004.

Playfield. This is the name for the (non-sprite) graphics that appear on the screen as a result of bitplane DMA. This name is a carry-over from the Atari 800 computer, and may be traced back further to the Atari VCS (now known as the Atari 2600) game system, all of which were designed by Jay Miner.

$DFF100 BPLCON0

Bitplane Control Register 0
Status: Write-Only. Chips: Agnus and Denise

Bit 1:	ERSY: External sync. This line is reset on power up. (1 = external; 0 = internal)
Bit 2:	LACE: Interlace enable. This line is reset on power up. (1 = interlaced)
Bit 3:	LPEN: Light pen enable. This line is reset on power up. (1 = enabled)
Bits 4-7:	Unused
Bit 8:	GAUD: Genlock audio enable (1 = enabled)
Bit 9:	COLOR: Composite video color enable (1 = color; 0 = black and white)
Bit 10:	DBLPF: Double (dual) playfield mode (1 = dual playfield)
Bit 11:	HOMOD: Hold-and-modify mode (1 = HAM)
Bits 12-14:	BPU: Bitplanes used
Bit 15:	HIRES: High resolution mode (1 = high resolution; 0 = low resolution)

Bit 1. 0 = Internal sync. 1 = External sync. The Amiga is able to accept an external signal for video synchronization. This is known as *genlocking*. Set this bit to 1 if you wish to genlock an external video source with Amiga graphics. Note: You'll need an external device known as a genlock to be able to use this mode. The genlocked video signal takes the place of playfield color 0.

Bit 2. 0 = Non-interlaced. 1 = Interlaced. The Amiga is capable of producing twice its normal vertical resolution with a mode known as *interlace*. In this mode, the Amiga displays two different images alternately. One image consists of the odd lines, the other of the even. This mode flickers considerably on a standard Amiga monitor. However, there are hardware devices available that take the two frames and combine them into a non-flickering image. Use of these devices requires a multisync monitor.

One advantage of interlace mode is that it works the same way as NTSC television. If you are using the Amiga in a video application (if you or genlocking or videotaping the Amiga's output, for example), you should set your Amiga to interlace mode for a better picture.

Bit 3. Built into the Amiga is the capacity to read a light pen. The light pen works by detecting when the video beam passes the tip of the pen. Using this information, you can figure out the horizontal and vertical position at which the light pen is directed. Set this bit to 1 if you want the light pen enabled, else set it to 0.

Bit 8. It has been alleged that audio data can be mixed on the BKGD pen during vertical blanking, and bit 8 enables this feature. In reality, however, this bit simply goes out on the pixel switch ZD during blanking periods.

Bit 9. On Amiga 1000s with color composite output, you can set this bit equal to 0 to output a monochrome display through the composite video port. This bit has no affect on the RGB output.

Bit 10. Set this bit to 1 if you wish to enable dual-playfield mode, or else set it to 0. When this bit is set, the odd bitplanes are grouped into one playfield (playfield 1) and the even ones are grouped into another (playfield 2). You can set which playfield appears on top of the other by setting bit 6 in BPLCON2 ($DFF106).

Bit 11. A value of 1 in this bit turns on HAM (Hold-and-modify) mode. HAM mode requires 5 or 6 bitplanes. HAM works in interlace, but not in hi-res.

Bits 12-14. Store the number of bitplanes used here:

000 = no bitplanes
001 = 1 bitplane. This allows 2 different playfield colors.
010 = 2 bitplanes. This allows 4 different playfield colors.
011 = 3 bitplanes. This allows 8 different playfield colors.
100 = 4 bitplanes. This allows 16 different playfield colors.
101 = 5 bitplanes. This allows 32 different colors. You cannot use this value with the HIRES flag (bit 15) turned on. You can use this value with HAM mode. See the description of the HAM flag (bit 11).
110 = 6 bitplanes. This allows 64 different colors, 32 of which are specified directly via the Amiga color registers and 32 of which are copies of the other 32, but with the red, green, and blue values each halved. This mode is known as Extra-HalfBrite (EHB), and it is invoked automatically when you choose six bitplanes. You cannot use this value with the HIRES flag (bit 15) turned on. You can use this value with HAM mode. See the description of the HAM flag (bit 11).

$DFF102 BPLCON1

Bitplane Control Register 1
Status: Write-Only. Chip: Denise

Bits 0-3: PF1H: Playfield 1 horizontal scroll.
Bits 4-7: PF2H: Playfield 2 horizontal scroll.

This register allows you to fine scroll the screen horizontally.

Vertical scrolling is trivial. To perform a vertical scroll, just increment or decrement the starting address of each bitplane by the width of a screen line in bytes.

Horizontal scrolling is trickier. You can perform a coarse (16 pixel) scroll by simply incrementing or decrementing the starting address of each bitplane by one word (two bytes), but to scroll at the pixel level you must use this register.

Four bits control the amount of fine scrolling for each playfield. For normal screens, set bits 0-3 and bits 4-7 to the same value. You should only set these groups of bits to different values for dual-playfield screens.

In low resolution, horizontal scrolling is in single-pixel increments. This translates to two-pixel increments in high resolution.

To set up a screen for horizontal scrolling, you must set up a screen in memory that is larger than the screen you are displaying. For example, you might have a horizontal scrolling game screen that is twice as wide as the screen you can see. Horizontal scrolling requires an extra word of data, so you must correctly set the data-fetch register

(DDFSTRT $DFF092) to include this extra word (subtract 8 from the register's normal value). You must set the modulo registers (BPLxMOD $DFF108) to the number of "extra" bytes on each line (those that would not exist if you weren't doing horizontal scrolling) minus two. For example, if you're displaying 40 bytes of an 80 byte wide screen, place the number 38 into the appropriate BPLxMOD registers.

$DFF104 BPLCON2

Bitplane Control Register 2

Status: Write-Only. Chip: Denise

This register allows you to determine priorities between the graphic elements that make up an Amiga screen, namely the eight sprites and two playfields.

Bits 0-2: PF1P: Playfield 1 vs. sprite priority
Bits 3-5: PF2P: Playfield 2 vs. sprite priority
Bit 6: PF2PRI: Playfield 1 vs. playfied 2 priority
(1 = playfield 2 is in front; 0 = playfield 1 is in front)

Bits 0-5. These bits allow to to specify whether sprites should pass in front of or behind a given set of bitplanes. The table below shows the allowable values. (Remember: Playfield 1 consists of the odd-numbered bitplanes and playfield 2 consists of the even-numbered bitplanes.)

bit 2	bit 1	bit 0	(for playfield 1)
bit 5	bit 4	bit 3	(for playfield 2)
0	0	0	Playfield has top priority
0	0	1	Playfield appears behind sprites 0-1
0	1	0	Playfield appears behind sprites 0-3
0	1	1	Playfield appears behind sprites 0-5
1	0	0	Playfield appears behind all sprites

Bit 6. This bit determines which playfield appears in front the other.

$DFF106 zilch

Unused

Location range: $DFF108-$DFF10A

Bitplane Modulo Registers

These registers control how many bytes are added to the bitplane pointers at the end of each video line. Normally, the value is zero. When the number is greater than zero, you

have an undisplayed segment of the screen to the right of the visible area. You can bring this area into view by incrementing the starting address of each bitplane. If you wish to make a smooth, pixel-by-pixel horizontal scroll, see BPLCON1 ($DFF102).

As an example, suppose the virtual screen you have in memory is 80 bytes wide, but the visible screen is only 40 bytes wide. You'll need to specify 40 an the modulo.

There are two bitplane modulo registers, one for each playfield. Unless you are using dual-playfield mode, set both of these registers to the same value.

Note: The lowest bit is ignored, so you must set these registers to an even value.

$DFF108 BPL1MOD

Bitplane Modulo 1
Status: Write-Only. Chip: Agnus

$DFF10A BPL2MOD

Bitplane Modulo 2
Status: Write-Only. Chip: Agnus

Location range: $DFF10C-$DFF10E

Unused

Location range: $DFF110-$DFF11A

Bitplane Data Registers

The Amiga hardware is currently able to display a maximum of six bitplanes. (If you look carefully, however, there are gaps between the hardware registers to allow two more bitplanes to be easily inserted. The Amiga system software allows for eight bitplanes also.) These bitplanes are combined to create a display based on the values stored in the BPLCON registers. Each bitplane has a corresponding data register. These registers are typically written to by bitplane DMA, but they may also be written to by the programmer directly.

These registers convert sixteen pixels worth of data from 1 to 6 bitplanes into a stream of serial data which is converted into a color (depending on the values in the color registers and the video mode) and output to the screen. In effect, what is occuring is a parallel to serial conversion. The bits are shifted out of the register to the left. Thus the uppermost bit corresponds to the leftmost pixel. Here's an example of how the conversion works, assuming a 5-bit non-HAM screen. The values stored in the registers are shown, then the corresponding stream of pixels is shown.

registers:
BPL1DAT 0000000110000000
BPL2DAT 0100010011000001
BPL3DAT 0110000010100010
BPL4DAT 0010000010000100
BPL5DAT 0000010011101000

pixels:
00000 (color register 0)
00110 (color register 6)
01100 (color register 12)
00000 (color register 0)
00000 (color register 0)
10010 (color register 18)
00000 (color register 0)
00001 (color register 1)
11111 (color register 31)
10010 (color register 18)
10100 (color register 20)
00000 (color register 0)
10000 (color register 16)
01000 (color register 8)
00100 (color register 4)
00010 (color register 2)

BPL1DAT is the trigger register. That means that you should write to BPL2DAT-BPL6DAT first, then write to BPL1DAT to trigger the parallel-to-serial conversion.

$DFF110 BPL1DAT

Bitplane Data Register 1
Status: Write-Only. Chip: Denise
Usually written to by bitplane DMA. This is the trigger register. As soon as this register is written to, the conversion from bitplane to pixel data begins.

A 1 bit in any of this register's bits contributes the value of 1 toward the corresponding pixel.

$DFF112 BPL2DAT

Bitplane Data Register 2
Status: Write-Only. Chip: Denise
Usually written to by bitplane DMA.

A 1 bit in any of this register's bits contributes the value of 2 toward the corresponding pixel.

$DFF114 BPL3DAT

Bitplane Data Register 3
Status: Write-Only. Chip: Denise
Usually written to by bitplane DMA.

A 1 bit in any of this register's bits contributes the value of 4 toward the corresponding pixel.

$DFF116 BPL4DAT

Bitplane Data Register 4
Status: Write-Only. Chip: Denise
Usually written to by bitplane DMA.

A 1 bit in any of this register's bits contributes the value of 8 toward the corresponding pixel.

$DFF118 BPL5DAT

Bitplane Data Register 5
Status: Write-Only. Chip: Denise
Usually written to by bitplane DMA.

A 1 bit in any of this register's bits contributes the value of 16 toward the corresponding pixel.

$DFF11A BPL6DAT

Bitplane Data Register 6
Status: Write-Only. Chip: Denise
Usually written to by bitplane DMA.

Bitplane 6 is only used in HAM and EHB modes.

Location range: $DFF11C-$DFF11E

Unused

Location Range: $DFF120-$DFF17E

Sprite Registers

The Amiga is best known for the animation capabilities of its blitter, but it also has powerful sprite hardware. The tricolor Intuition mouse pointer is an Amiga sprite at its simplest. Sprites are capable of much more than you may realize. For instance, the same sprite may appear several times on the same screen. In addition, sprites may be "attached" to make them more colorful.

Sprites have some advantages over the blitter. While the blitter must render its images directly into display memory, sprites use a separate area of memory. Sprites are rendered "on the fly" as the screen is displayed.

Programmers who use the operating system have two choices when it comes to using sprites. First, they can use the SimpleSprite structure, which roughly corresponds to a hardware sprite. Second, they can use a VSprite, which can handle an arbitrary number of sprites. Of course, the operating system must figure out how best to map the arbitrary number of sprites into the eight physical sprites. If you've played *Marble Madness,* you may have noticed that the marbles and other objects change color or disappear occasionally. This is a common problem with VSprites. Therefore, most programmers prefer to take direct control over the sprites, either by using the operating system's SimpleSprite structure and related calls, or by writing to the sprite hardware under Copper or direct program control.

There are eight sprites. Each sprite is 16 pixels wide and an arbitrary size high. The pixels are low resolution, even when displayed in high resolution and/or interlace modes. Each pixel of a sprite can be one of three colors or transparent.

Sprites 0 and 1 share color registers 17, 18, and 19. Sprites 2 and 3 share color registers 21, 22, and 23. Sprites 4 and 5 share color registers 25, 26, and 27. And sprites 6 and 7 share color registers 29, 30, and 31. If your screen uses 4 bitplanes or less, you won't have to worry about sprites sharing the color registers with the playfield. However, if you have a 5 bitplane (32 color) or 6 bitplane (64 color) screen, you'll have to plan your color choices carefully; 6 bitplane HAM screens use only the first 16 color registers, so sprites work nicely on HAM screens.

If you desire more colorful sprites, you can create four sprites by combining sprites 0 and 1, 2 and 3, 4 and 5, and 6 and 7. Each of the composite sprites will use color registers 17 through 31.

Sprite priorities are fixed on the Amiga. Sprite 1 always is always obscured by sprite 0. Sprite 2 is always obscured by sprites 0 and 1, and so on. Sprite priorities relative to the playfield can be assigned. See the bitplane control registers $DFF100-$DFF106.

$DFF120 SPROPT (SPROPTH = $DFF120 and SPROPTL = $DFF122)

Sprite Pointer 0

Status: Write-Only. Chip: Agnus

This register holds the address of the Sprite 0 DMA data. This address must be in chip RAM, therefore it is 18 bits long on Amigas that have 512K of chip RAM and 19 bits on Amigas that have 1 meg of chip RAM.

Although you do not absolutely need to rely on sprite DMA, it's probably the best bet for 90 percent of sprite applications. If fact, it's hard to imagine an instance where you would want to take over the responsibilities that the Amiga hardware can easily perform.

This register acts as a DMA counter, so you must initialize it during every vertical blank. You can either do this within a copper list or within a vertical blank interrupt server.

The address which you set this register to should contain valid sprite data in chip RAM. Use this format (each value is 32 bits):

Sprite Control Value
Image Data for line 1
Image Data for line 2
.
.
.
Image Data for last line

Repeat this format for each occurance of the sprite on the screen. For instance, you might have the same sprite engine used in three different vertical areas of the screen. In this case, you'd have the above format three times.

At the end of the data, store a $00000000.

Here is an example of a sprite adapted from the Amiga Hardware Reference Manual.

```
SPRITE:
        DC.L    $6D607200       ;VSTART, HSTART, and VSTOP
        DC.L    $099007E0       ;Sprite data
        DC.L    $13C80FF0       ;Sprite data
        DC.L    $23C41FF8       ;Sprite data
        DC.L    $13C80FF0       ;Sprite data
        DC.L    $099007E0       ;Sprite data
        DC.L    $00000000       ;end of structure
```

The Amiga is often said to have reusable sprite engines. This means that the Amiga can display the same sprite in several different places. There is one significant limitation to this ability: You must ensure there is at least one blank line vertically between two instances of the same sprite. For example, if you have a sprite that stops displaying on vertical line 25, you can display a second one beginning on line 27 or later.

$DFF124 SPR1PT ($DFF124 and SPR1PTL = $DFF126)
Sprite Pointer 1
Status: Write-Only. Chip: Agnus

$DFF128 SPR2PT ($DFF128 and SPR2PTL = $DFF12A)
Sprite Pointer 2
Status: Write-Only. Chip: Agnus

$DFF12C SPR3PT ($DFF12C and SPR3PTL = $DFF12E)
Sprite Pointer 3
Status: Write-Only. Chip: Agnus

$DFF130 SPR4PT ($DFF130 and SPR4PTL = $DFF132)
Sprite Pointer 4
Status: Write-Only. Chip: Agnus

$DFF134 SPR5PT ($DFF134 and SPR5PTL = $DFF136)
Sprite Pointer 5
Status: Write-Only. Chip: Agnus

$DFF138 SPR6PT ($DFF138 and SPR6PTL = $DFF13A)
Sprite Pointer 6
Status: Write-Only. Chip: Agnus

$DFF13C SPR7PT ($DFF13C and SPR7PTL = $DFF13E)
Sprite Pointer 7
Status: Write-Only. Chip: Agnus

$DFF140 SPR0POS ($DFF140 and SPR0CTL = $DFF142
Sprite Position 0 and Sprite Control 0
Status: Write-Only. Chip: Agnus and Denise

These two 16-bit registers can be thought of as a single 32-bit register, which holds information about how sprite 0 is to be displayed. Information included is vertical and horizontal starting locations, vertical stopping location, and whether this sprite should be attached to its companion (0 and 1 are companions, 2 and 3 are companions, and so forth).

The Sprite Control Value consists of the following bits, the numbers of which should be stored in the:

Bit 0:	HSTART lowest bit
Bit 1:	VSTOP highest bit
Bit 2:	VSTART highest bit
Bits 3-6:	Always set to 0
Bit 7:	Attachment bit. (1 = attached; 0 = not attached)
Bits 8-15:	The lowest 8 bits of VSTOP
Bits 16-23:	The highest 8 bits of HSTART
Bits 24-31:	The lowest 8 bits of VSTART

Note that HSTART, VSTART, and VSTOP are all 9-bit values, but they are broken up awkwardly among the bits. A mathematical formula for the value to place in the register might look like this:

(VSTART MOD 2) + ATTACH * 128 + (HSTART DIV 2) * 256 +(VSTOP DIV 2) * 65536 + (VSTOP MOD 256) * 67108864 +(HSTART DIV 256) * 13417728 + (VSTART MOD 256) * 268435456

where MOD stands for the remainder of an integer division and DIV stands for the quotient of an integer division. This equation is quite cumbersome. You'll find that shifting and ORing leads to a more elegant method of setting this value. Note that the lowest 8 bits of the VSTOP and VSTART and the highest 8 bits of HSTART are located on byte boundaries. It is sometimes convenient to ignore the other bits, although this solution restricts the vertical position and makes horizontal positioning coarser.

The HSTART value is your sprite's intended horizontal position. Typically, the horizontal position 0 is 64 pixels to the left of the screen. This number can change depending on your preferences settings. Suppose you want to position a sprite at the horizontal position which corresponds to pixel 10. In this case, you'll add 10 to 64 to get 74. Place the number 74 in for HSTART in the above equation. If you're setting up your own copper list, you'll know the value to add to your desired offset. If you're using a screen set up by the screen, you can find the offset in the View structure.

The VSTART value is your sprite's intended vertical position. Typically, the vertical position 0 is 44 pixels above the top line of the screen. This number can change depending on your preferences settings. Suppose you want to position a sprite at the vertical position which corresponds to pixel 12. In this case, you'll add 12 to 44 to get 56. Place the number 56 in for VSTART in the above equation. If you're setting up your own copper list, you'll know the value to add to your desired offset. If you're using a screen set up by the screen, you can find the offset in the View structure.

The VSTOP value is the line on which you want your sprite to stop displaying. Simply add the height of the sprite to VSTART to get this value.

This register's attachment bit (bit 7) is ignored in even-numbered sprites. For example, if you wish to attach sprites 0 and 1, set the attachment bit in sprite 1. When you attach two sprites, set the HSTART, VSTART, and VSTOP of the two sprites to the same values.

Keep in mind that you can think of this location as either a single 32-bit register or a pair of 16-bit registers. If you wish to write to this location from within a Copper list, you'll need to do it with two MOVE instructions, since a MOVE instruction requires a WORD-length operand.

This location is usually not written to by the programmer. Instead, it is written to by sprite DMA.

$DFF144 SPR0DAT (SPR0DATA = $DFF144 and SPR0DATB = $DFF146)

Sprite Data Register 0

Status: Write-Only. Chip: Denise

These two 16-bit registers can be thought of as a single 32-bit register, which holds information about the pixels which sprite 0 is to display.

Sprites are two bitplanes deep. That means four different colors can be represented. SPR0DATA is considered the low plane and SPR0DATB is considered the high plane. That means SPR0DATA can contribute a value of 0 or 1 toward the final color and SPR0DATB can contribute a value of 0 or 2 toward the final color. Here's how it works.

If SPR0DATA receives the binary value 0011001100001111 and SPR0DATB receives the binary value 0101010100110011, then the pixels which are written out for sprite 0 will be 0213021300221133. This is easier to see if we replace the 1s in SPR0DATB with 2s, since the 1 bits in SPR0DATB contribute a 2 to the final pixel color value.

0011001100001111 (SPR0DATA)
0202020200220022 (SPR0DATB)
0213021300221133 (pixels)

Sprites are designed to move over backgrounds, so the hardware designers decided that a 0 pixel would let the background show through. This means that a sprite made up of all 0s would be completely invisible.

Sprite 0 shares sprite 1's color registers. The usage is as follows:

Pixel Value 0 = transparent
Pixel Value 1 = color register 17
Pixel Value 2 = color register 18
Pixel Value 3 = color register 19

Note that sprites are 16 pixels wide—always. There is no way to make a sprite any wider. The pixels are low resolution noninterlaced pixels. If you want an object that is more than 16 pixels, you must use more than one sprite or forget about sprites and use blitter objects instead.

By setting the ATTACH bit in the SPR0POS register, you can join sprites 0 and 1 in order to utilize color registers 17-31. Colors work much the same as in the case of the unattached sprite, but instead of just two words of data determining the color, four words of data are used. Here's an example similar to the one above. I'll assume that you're joining sprites 0 and 1.

0 0 1 1 0 0 1 1 0 0 0 0 1 1 1 1 (SPR0DATA)
0 2 0 2 0 2 0 2 0 0 2 2 0 0 2 2 (SPR0DATB)
0 0 0 0 0 0 0 0 4 4 4 4 4 4 4 4 (SPR1DATA)
0 0 0 0 8 8 8 8 0 0 0 0 8 8 8 8 (SPR1DATB)
0 2 1 3 0 2 9 B 4 4 6 6 D D F F (pixels)

As you can see, the odd-numbered sprite's bitplanes are considered to be of higher order than the even-numbered sprite's.

Keep in mind that you can think of this location as either a single 32-bit register or a pair of 16-bit registers. If you wish to write to this location from within a copper list, you'll need to do it with two MOVE instructions, since a MOVE instruction requires a word-length operand.

This location is usually not written to by the programmer. Instead, it is written to by sprite DMA. If you write to it directly, write to SPR0DATB first and SPR0DATA

second. The data will then be displayed on every line at the horizontal position specified in SPR0POS and SPR0CTL until you write to SPR0DATA again. SPR0DATA acts as a switch to turn the sprite on. SPR0CTL acts as the corresponding off switch.

$DFF148 SPR1POS (SPR1POS = $DFF148 and SPR1CTL = $DFF14A)

Sprite Position 1 and Sprite Control 1
Status: Write-Only. Chip: Agnus and Denise
The sprite control register for sprite 1. See location $DFF140.

$DFF14C SPR1DAT (SPR1DATA = $DFF14C and SPR1DATB = $DFF14E)

Sprite Data Register 1
Status: Write-Only. Chip: Denise
Sprite data for sprite 1. See location $DFF144.

Sprite 1 shares sprite 0's color registers. The usage is as follows:

Pixel Value 0 = transparent
Pixel Value 1 = color register 17
Pixel Value 2 = color register 18
Pixel Value 3 = color register 19

$DFF150 SPR2POS (SPR2POS = $DFF150 and SPR1CTL = $DFF152)

Sprite Position 2 and Sprite Control 2
Status: Write-Only. Chip: Agnus and Denise
The sprite control register for sprite 2. See location $DFF140.

$DFF154 SPR2DAT (SPR2DATA = $DFF154 and SPR2DATB = $DFF156)

Sprite Data Register 1
Status: Write-Only. Chip: Denise
Sprite data for sprite 2. See location $DFF144.

Sprite 2 shares sprite 3's color registers. The usage is as follows:

Pixel Value 0 = transparent
Pixel Value 1 = color register 21
Pixel Value 2 = color register 22
Pixel Value 3 = color register 23

$DFF158 SPR3POS (SPR3POS = $DFF158 and SPR3CTL = $DFF15A)

Sprite Position 2 and Sprite Control 2
Status: Write-Only. Chip: Agnus and Denise
The sprite control register for sprite 3. See location $DFF140.

$DFF15C SPR3DAT (SPR3DATA = $DFF15C and SPR3DATB = $DFF15E)

Sprite Data Register 3
Status: Write-Only. Chip: Denise
Sprite data for sprite 3. See location $DFF144.

Sprite 3 shares sprite 2's color registers. The usage is as follows:

Pixel Value 0 = transparent
Pixel Value 1 = color register 21
Pixel Value 2 = color register 22
Pixel Value 3 = color register 23

$DFF160 SPR4POS (SPR4POS = $DFF160 and SPR4CTL = $DFF162)

Sprite Position 3 and Sprite Control 3
Status: Write-Only. Chip: Agnus and Denise
The sprite control register for sprite 4. See location $DFF140.

$DFF164 SPR4DAT (SPR4DATA = $DFF164 and SPR4DATB = $DFF166)

Sprite Data Register 4
Status: Write-Only. Chip: Denise
Sprite data for sprite 4. See location $DFF144.

Sprite 4 shares sprite 5's color registers. The usage is as follows:

Pixel Value 0 = transparent
Pixel Value 1 = color register 25
Pixel Value 2 = color register 26
Pixel Value 3 = color register 27

$DFF168 SPR5POS (SPR5POS = $DFF168 and SPR5CTL = $DFF16A)

Sprite Position 5 and Sprite Control 5
Status: Write-Only. Chip: Agnus and Denise
The sprite control register for sprite 5. See location $DFF140.

$DFF16C SPR4DAT (SPR5DATA = $DFF16C and SPR5DATB = $DFF16E)

Sprite Data Register 5
Status: Write-Only. Chip: Denise
Sprite data for sprite 5. See location $DFF144.

Sprite 5 shares sprite 4's color registers. The usage is as follows:

Pixel Value 0 = transparent
Pixel Value 1 = color register 25
Pixel Value 2 = color register 26
Pixel Value 3 = color register 27

$DFF170 SPR5POS (SPR5POS = $DFF170 and SPR5CTL = $DFF172)

Sprite Position 5 and Sprite Control 5
Status: Write-Only. Chip: Agnus and Denise
The sprite control register for sprite 6. See location $DFF140.

$DFF174 SPR6DAT (SPR6DATA = $DFF174 and SPR6DATB = $DFF176)

Sprite Data Register 6
Status: Write-Only. Chip: Denise
Sprite data for sprite 6. See location $DFF144.

Sprite 6 shares sprite 7's color registers. The usage is as follows:

Pixel Value 0 = transparent
Pixel Value 1 = color register 29
Pixel Value 2 = color register 30
Pixel Value 3 = color register 31

$DFF178 SPR7POS (SPR7POS = $DFF178 and SPR7CTL = $DFF17A)

Sprite Position 7 and Sprite Control 7
Status: Write-Only. Chip: Agnus and Denise
The sprite control register for sprite 7. See location $DFF140.

$DFF17C SPR7DAT (SPR7DATA = $DFF17C and SPR7DATB = $DFF17E)

Sprite Data Register 7
Status: Write-Only. Chip: Denise
Sprite data for sprite 7. See location $DFF144.

Sprite 7 shares sprite 6's color registers. The usage is as follows:

Pixel Value 0 = transparent
Pixel Value 1 = color register 29
Pixel Value 2 = color register 30
Pixel Value 3 = color register 31

Location Range: $DFF180-$DFF1BE

Color Registers
The Amiga has 32 color registers. That means that, under normal operations, a maximum of 32 colors can be displayed on the screen. These colors are chosen by pixel values 0-31. In Extra HalfBrite and HAM modes, more colors are displayed via certain tricks. See the BPLCON0 ($DFF100) for more details. You can also obtain more colors by changing these colors on the fly from within the copper list. See the description of the copper registers ($DFF080-$DFF08C) for more information.

Each color register consists of 12 bits of useful information (the upper four bits are ignored); 12 bits of color information mean that 4096 different colors can be chosen from. That means the Amiga has a palette of 4096 colors. The registers are set up as follows:
Bits 0-3: The blue intensity
This number can range from 0 (no blue) to 15 (full blue)
Bits 4-7: The green intensity
This number can range from 0 (no green) to 15 (full green)
Bits 8-11: The red intensity
This number can range from 0 (no red) to 15 (full red)
Bits 12-15: Unused

This setup has many implications that should be considered when you choose colors. There are seven colors that can be represented in 16 different intensities. Those colors are gray, red, green, blue, yellow, magenta, and cyan. Here is a table for those colors. Colors are represented here as hexadecimal values. Given the Amiga's color register layout, this lets you quickly see the red, green, and blue intensities that make up the color.

Intensity	Gray	Red	Green	Blue	Yellow	Magenta	Cyan
0	000	000	000	000	000	000	000
1	111	100	010	001	110	101	011
2	222	200	020	002	220	202	022
3	333	300	030	003	330	303	033
4	444	400	040	004	440	404	044
5	555	500	050	005	550	505	055
6	666	600	060	006	660	606	066
7	777	700	070	007	770	707	077
8	888	800	080	008	880	808	088
9	999	900	090	009	990	909	099
10	AAA	A00	0A0	00A	AA0	A0A	0AA
1	BBB	B00	0B0	00B	BB0	B0B	0BB
12	CCC	C00	0C0	00C	CC0	C0C	0CC
13	DDD	D00	0D0	00D	DD0	D0D	0DD
14	EEE	E00	0E0	00E	EE0	E0E	0EE
15	FFF	F00	0F0	00F	FF0	F0F	0FF

Six colors can be reproduced in 8 levels—for example, orange, which consists of two parts red and one part green. The eight levels of orange are $000, $210, $420, $630, $840, $A50, $C60, and $E70.

Other colors are limited to even fewer levels, such as the extreme case of an slightly orangish red ($F10), which has only two intensity levels—$F10 and $000.

These limitations are important. For example, you cannot display a 32-shade monochrome screen without resorting to dithering or approximations. Also, if you "fade-in" a screen from black to full-color by ramping up the color registers from $000 to their full-color values, you'll notice that some colors make the transition more smoothly than others.

Here is a table of common colors.

Color	RGB value
Black	000
Blue	00F
Green	0F0
Cyan	0FF
Red	F00
Purple	F0F
Orange	E70
Yellow	FF0
White	FFF
Light Gray	CCC
Medium Gray	888
Dark Gray	444

$DFF180 COLOR00

Color Register 0
Status: Write-Only. Chip: Denise
This color is displayed when the playfield (bitmap) pixel value is 0 (binary 00000). This color is also displayed around the borders of the screen. If genlocking is on (see BPLCON0 $DFF100), the source video is displayed instead. This color can be visible on all screens.

$DFF182 COLOR01

Color Register 1
Status: Write-Only. Chip: Denise
This color is displayed when the playfield (bitmap) pixel number is 1 (binary 00001). This color can be visible on screens with one or more bitplanes.

$DFF184 COLOR02

Color Register 2
Status: Write-Only. Chip: Denise
This color is displayed when the playfield (bitmap) pixel number is 2 (binary 00010). This color can be visible on screens with two or more bitplanes.

$DFF186 COLOR03

Color Register 3
Status: Write-Only. Chip: Denise
This color is displayed when the playfield (bitmap) pixel number is 3 (binary 00011). This color can be visible on screens with two or more bitplanes.

$DFF188 COLOR04

Color Register 4
Status: Write-Only. Chip: Denise
This color is displayed when the playfield (bitmap) pixel number is 4 (binary 00100). This color can be visible on screens with three or more bitplanes.

$DFF18A COLOR05

Color Register 5
Status: Write-Only. Chip: Denise
This color is displayed when the playfield (bitmap) pixel number is 5 (binary 00101). This color can be visible on screens with three or more bitplanes.

$DFF18C COLOR06
Color Register 6
Status: Write-Only. Chip: Denise
This color is displayed when the playfield (bitmap) pixel number is 6 (binary 00110). This color can be visible on screens with three or more bitplanes.

$DFF18E COLOR07
Color Register 7
Status: Write-Only. Chip: Denise
This color is displayed when the playfield (bitmap) pixel number is 7 (binary 00111). This color can be visible on screens with three or more bitplanes.

$DFF190 COLOR08
Color Register 8
Status: Write-Only. Chip: Denise
This color is displayed when the playfield (bitmap) pixel number is 8 (binary 01000). This color can be visible on screens with four or more bitplanes.

$DFF192 COLOR09
Color Register 9
Status: Write-Only. Chip: Denise
This color is displayed when the playfield (bitmap) pixel number is 9 (binary 01001), or in dual-playfield mode when playfield 2's pixel number is 1 (binary 001). This color can be visible on screens with four or more bitplanes.

$DFF194 COLOR10
Color Register 10
Status: Write-Only. Chip: Denise
This color is displayed when the playfield (bitmap) pixel number is 10 (binary 01010), or in dual-playfield mode when playfield 2's pixel number is 2 (binary 010). This color can be visible on screens with four or more bitplanes.

$DFF196 COLOR11
Color Register 11
Status: Write-Only. Chip: Denise
This color is displayed when the playfield (bitmap) pixel number is 11 (binary 01011), or in dual-playfield mode when playfield 2's pixel number is 3 (binary 011). This color can be visible on screens with four or more bitplanes.

$DFF198 COLOR12

Color Register 12
Status: Write-Only. Chip: Denise
This color is displayed when the playfield (bitmap) pixel number is 12 (binary 01100), or in dual-playfield mode when playfield 2's pixel number is 4 (binary 100). This color can be visible on screens with four or more bitplanes.

$DFF19A COLOR13

Color Register 13
Status: Write-Only. Chip: Denise
This color is displayed when the playfield (bitmap) pixel number is 13 (binary 01101), or in dual-playfield mode when playfield 2's pixel number is 5 (binary 101). This color can be visible on screens with four or more bitplanes.

$DFF19C COLOR14

Color Register 14
Status: Write-Only. Chip: Denise
This color is displayed when the playfield (bitmap) pixel number is 14 (binary 01110), or in dual-playfield mode when playfield 2's pixel number is 6 (binary 110). This color can be visible on screens with four or more bitplanes.

$DFF19E COLOR15

Color Register 15
Status: Write-Only. Chip: Denise
This color is displayed when the playfield (bitmap) pixel number is 15 (binary 01111), or in dual-playfield mode when playfield 2's pixel number is 7 (binary 111). This color can be visible on screens with four or more bitplanes.

$DFF1A0 COLOR16

Color Register 16
Status: Write-Only. Chip: Denise
This color is displayed when the playfield (bitmap) pixel number is 16 (binary 10000). This color can be visible on screens with five or more bitplanes.

$DFF1A2 COLOR17

Color Register 17
Status: Write-Only. Chip: Denise
This color is displayed when the playfield (bitmap) pixel number is 17 (binary 10001),

or when sprites 0 or 1 use their first color (binary 01), or when two sprites are joined and they use their first color (binary 0001). This color can be visible on screens with five or more bitplanes or on a screen that uses sprites.

$DFF1A4 COLOR18

Color Register 18

Status: Write-Only. Chip: Denise

This color is displayed when the playfield (bitmap) pixel number is 18 (binary 10010), or when sprites 0 or 1 use their second color (binary 10), or when two sprites are joined and they use their second color (binary 0010). This color can be visible on screens with five or more bitplanes or on a screen that uses sprites.

$DFF1A6 COLOR19

Color Register 19

Status: Write-Only. Chip: Denise

This color is displayed when the playfield (bitmap) pixel number is 19 (binary 10011), or when sprites 0 or 1 use their third color (binary 11), or when two sprites are joined and they use their third color (binary 0011). This color can be visible on screens with five or more bitplanes or on a screen that uses sprites.

$DFF1A8 COLOR20

Color Register 20

Status: Write-Only. Chip: Denise

This color is displayed when the playfield (bitmap) pixel number is 20 (binary 10100), or when two sprites are joined and they use their fourth color (binary 0100). This color can be visible on screens with five or more bitplanes or on a screen that uses sprites.

$DFF1AA COLOR21

Color Register 21

Status: Write-Only. Chip: Denise

This color is displayed when the playfield (bitmap) pixel number is 21 (binary 10101), or when sprites 2 or 3 use their first color (binary 01), or when two sprites are joined and they use their fifth color (binary 0101). This color can be visible on screens with five or more bitplanes or on a screen that uses sprites.

$DFF1AC COLOR22

Color Register 22
Status: Write-Only. Chip: Denise
This color is displayed when the playfield (bitmap) pixel number is 22 (binary 10110), or when sprites 2 or 3 use their second color (binary 10), or when two sprites are joined and they use their sixth color (binary 0110). This color can be visible on screens with five or more bitplanes or on a screen that uses sprites.

$DFF1AE COLOR23

Color Register 23
Status: Write-Only. Chip: Denise
This color is displayed when the playfield (bitmap) pixel number is 23 (binary 10111), or when sprites 2 or 3 use their third color (binary 11), or when two sprites are joined and they use their seventh color (binary 0111). This color can be visible on screens with five or more bitplanes or on a screen that uses sprites.

$DFF1B0 COLOR24

Color Register 24
Status: Write-Only. Chip: Denise
This color is displayed when the playfield (bitmap) pixel number is 24 (binary 11000), or when two sprites are joined and they use their eighth color (binary 1000). This color can be visible on screens with five or more bitplanes or on a screen that uses sprites.

$DFF1B2 COLOR25

Color Register 25
Status: Write-Only. Chip: Denise
This color is displayed when the playfield (bitmap) pixel number is 25 (binary 11001), or when sprites 4 or 5 use their first color (binary 01), or when two sprites are joined and they use their ninth color (binary 1001). This color can be visible on screens with five or more bitplanes or on a screen that uses sprites.

$DFF1B4 COLOR26

Color Register 26
Status: Write-Only. Chip: Denise
This color is displayed when the playfield (bitmap) pixel number is 26 (binary 11010), or when sprites 4 or 5 use their second color (binary 10), or when two sprites are joined and they use their tenth color (binary 1010). This color can be visible on screens with five or more bitplanes or on a screen that uses sprites.

$DFF1B6 COLOR27

Color Register 27

Status: Write-Only. Chip: Denise

This color is displayed when the playfield (bitmap) pixel number is 27 (binary 11011), or when sprites 4 or 5 use their third color (binary 11), or when two sprites are joined and they use their eleventh color (binary 1011). This color can be visible on screens with five or more bitplanes or on a screen that uses sprites.

$DFF1B8 COLOR28

Color Register 28

Status: Write-Only. Chip: Denise

This color is displayed when the playfield (bitmap) pixel number is 28 (binary 11100), or when two sprites are joined and they use their twelfth color (binary 1100). This color can be visible on screens with five or more bitplanes or on a screen that uses sprites.

$DFF1BA COLOR29

Color Register 29

Status: Write-Only. Chip: Denise

This color is displayed when the playfield (bitmap) pixel number is 29 (binary 11101), or when sprites 6 or 7 use their first color (binary 01), or when two sprites are joined and they use their thirteenth color (binary 1101). This color can be visible on screens with five or more bitplanes or on a screen that uses sprites.

$DFF1BC COLOR30

Color Register 30

Status: Write-Only. Chip: Denise

This color is displayed when the playfield (bitmap) pixel number is 30 (binary 11110), or when sprites 6 or 7 use their second color (binary 10), or when two sprites are joined and they use their fourteenth color (binary 1110). This color can be visible on screens with five or more bitplanes or on a screen that uses sprites.

$DFF1BE COLOR31

Color Register 31

Status: Write-Only. Chip: Denise

This color is displayed when the playfield (bitmap) pixel number is 31 (binary 11111), or when sprites 6 or 7 use their third color (binary 10), or when two sprites are joined and they use their fifteenth color (binary 1111). This color can be visible on screens with five or more bitplanes or on a screen that uses sprites.

8520 CIA Chips

Location Range: $BFD000-$BFDF00

CIA-B

Locations $BFD000-$BFDF00 are used to address the Amiga's second Complex Interface Adapter (CIA-B). Since the chip itself is identical to CIA-A, which is addressed at $BFE001, the discussion here will be limited to the use the Amiga makes of this particular chip. For more general information on the chip registers, please see the corresponding entries for CIA-A.

Location Range: $BFD000-$BFD300

CIA-B Data Ports A and B

The main differences between the CIA-A and CIA-B chips is the use to which Data Ports A and B are put. Data Port A on CIA-B controls the status lines found on the Amiga's parallel and serial ports. Data Port B provides direct control over the Amiga's floppy disk controller.

$BFD100 PRA

Peripheral Data Register for Data Port A
Status: Read/Write. Chip: CIA-B

Bit 0: BUSY: Centronics busy (parallel port pin 11)
Bit 1: POUT: Centronics paper out (parallel port pin 12)
Bit 2: SEL: Centronics select (parallel port pin 13)
Bit 3: DSR: RS-232 data set ready (serial port pin 6)
Bit 4: CTS: RS-232 clear to send (serial port pin 5)
Bit 5: DCD: RS-232 carrier detect (serial port pin 8)
Bit 6: RTS: RS-232 request to send (serial port pin 4)
Bit 7: DTR: RS-232 data terminal ready (serial port pin 20)

This register is used to control the status lines found on the Amiga's serial and parallel ports. Although each bit is normally used for either input or output, exclusively, it is possible to both input and output data on any of these lines.

$BFD100 PRB

Peripheral Data Register for Data Port B
Status: Read/Write. Chip: CIA-B

Bit 0: STEP: Move drive head by one track in direction specified by the DIR bit (set to 1, 0, and then 1 again to move head)
Bit 1: DIR: Direction to move drive head (1 = outward; 0 = inward)
Bit 2: SIDE: Select drive head (1 = bottom; 0 = top)
Bit 3: SEL0: Select DF0: (1 = not selected; 0 = selected)
Bit 4: SEL1: Select DF1: (1 = not selected; 0 = selected)
Bit 5: SEL2: Select DF2: (1 = not selected; 0 = selected)
Bit 6: SEL3: Select DF3: (1 = not selected; 0 = selected)
Bit 7: MTR: Motor on-off status (1 = motor off; 0 = motor on)

Data Port B has direct control over the Amiga's floppy disk drives. Approach this register with extreme caution. Many floppy disk drives will attempt to do whatever you tell them to, even if that means jamming the drive's head beyond track 0 (definitely *not* a recommended procedure). And before you even consider using this register for performing low-level disk access, you must disable AmigaDOS and take complete control of the computer.

Bit 0. This bit controls the movement of the drive head for all selected drives. To move the drive head, you must toggle the value of this bit from 1 to 0, and then back to 1 again. One such operation moves the drive head the distance of one track.

Before moving the drive head, you must select a direction: either outward towards track 0, or inwards towards track 79. Bit 1 of this register determines the direction that the drive heads will move.

After moving the drive head, it's very important that you wait at least three milliseconds before performing any other disk activities. Because software timing loops are inaccurate (they depend too much on the computer's clock speed, which may vary from computer to computer), it's recommended that you use one of the CIA chip's timers to pause the required three milliseconds.

Bit 1. The value of this bit determines the direction in which the selected drives' heads will move when bit 0 of this register is changed from high (1) to low (0). To move the drives' heads out towards track 0, store a 1 here. To move the drives' heads in towards track 79, store a 0 here.

To be sure of the direction that you are moving, always set this register prior to toggling bit 0, and *never* attempt to move a drive head in further than track 79 or out past track 0. (You can check to see if the drive head of a selected drive is on track 0 by reading bit 4 of the CIA-A chip's PRA register, location $BFE001).

Bit 2. Amiga floppy disks are double sided. That means that the disk drives must have two drive heads—one for the top side of the disk and one for the bottom side of the disk. This bit determines which drive head will be used when it comes time to read and/or write data. To select the bottom drive head, store a 1 here. To select the top drive head, store a 0 here. As with the previous bits, the value of this bit only affects the currently selected drives.

Bits 3-6. These four bits allow you to choose which of the Amiga's floppy disk drives are selected. Only selected drives are affected by the values stored in the register's remaining bits.

The Amiga's hardware supports up to four 3 1/2-inch floppy disk drives. On the Amiga 500 and 1000, the internal drive is always known as drive 0 (DF0:). External drives are daisy chained together. The drive connected directly to the computer is drive 1, the drive connected to drive 1 is drive 2, and the drive connected to drive 2 is drive 3. With the Amiga 2000, 2500, and 3000, the two internal drives make up drives 0 and 1; the first external drive is drive 2 (DF2:), even if only one internal drive is present. Any drive connected to this external drive is drive 3.

To select a disk drive, set its corresponding bit equal to 0. To deselect a drive, set its corresponding bit equal to 1. Bits 3, 4, 5, and 6 correspond to drives 0, 1, 2, and 3, respectively. Any combination of drives may be selected at any one time. The other bits in this register affect all selected drives, so it's possible to perform such tasks as simultaneously moving the drive heads on more than one drive.

Bit 7. This bit turns on and off the motors of the selected drives. If you store a 1 here, the motors are turned off. If you store a 0 here, the motors are turned on. The on-off state of a motor is reflected by the drive light found on the front of the disk drive.

This bit should be set *before* you select a drive. If a drive is already selected, and you wish to change the state of its motor, you should deselect the drive, set this bit, and then reselect the desired drive.

When turning on a drive's motor, you must wait for the motor to reach its full rotation speed before performing any other disk activities. Bit 5 of the CIA-A chip's PRA register (see location $BFE001) is set equal 0 whenever the drive has reached full speed and is ready to receive another command.

$BFD200 DDRA

Data Direction Register A
Status: Read/Write. Chip: CIA-B

Bit 0: Select bit 0 of PRA for input or output.
Normally set to 0 (0 = input; 1 = output)

Bit 1: Select bit 1 of PRA for input or output.
Normally set to 0 (0 = input; 1 = output)
Bit 2: Select bit 2 of PRA for input or output.
Normally set to 0 (0 = input; 1 = output)
Bit 3: Select bit 3 of PRA for input or output.
Normally set to 0 (0 = input; 1 = output)
Bit 4: Select bit 4 of PRA for input or output.
Normally set to 0 (0 = input; 1 = output)
Bit 5: Select bit 5 of PRA for input or output.
Normally set to 0 (0 = input; 1 = output)
Bit 6: Select bit 6 of PRA for input or output.
Normally set to 1 (0 = input; 1 = output)
Bit 7: Select bit 7 of PRA for input or output.
Normally set to 1 (0 = input; 1 = output)

$BFD300 DDRB

Data Direction Register B
Status: Read/Write. Chip: CIA-B

Bit 0: Select bit 0 of PRB for input or output.
Must be set to 1 (0 = input; 1 = output)
Bit 1: Select bit 1 of PRB for input or output.
Must be set to 1 (0 = input; 1 = output)
Bit 2: Select bit 2 of PRB for input or output.
Must be set to 1 (0 = input; 1 = output)
Bit 3: Select bit 3 of PRB for input or output.
Must be set to 1 (0 = input; 1 = output)
Bit 4: Select bit 4 of PRB for input or output.
Must be set to 1 (0 = input; 1 = output)
Bit 5: Select bit 5 of PRB for input or output.
Must be set to 1 (0 = input; 1 = output)
Bit 6: Select bit 6 of PRB for input or output.
Must be set to 1 (0 = input; 1 = output)
Bit 7: Select bit 7 of PRB for input or output.
Must be set to 1 (0 = input; 1 = output)

Location Range: $BFD400-$BFD700

CIA-B Timers A and B Low and High Bytes

According to some early Commodore-Amiga manuals, Timer A is reserved for Commodore 8-bit serial-bus communications—the bus system used by the Commodore 64 to talk to 1541, 1571, and 1581 disk drives as well as Commodore 64-compatible printers. For all practical purposes, however, the Amiga has no use for this timer. Feel free to make use of it yourself. Timer B is also ignored by the Amiga's operating system, so feel free to use it as well.

For more details on the operation of these registers, see the entry for the CIA-A Timer A and Timer B registers at $BFE401-$BFE701.

$BFD400 TALO

Timer A Low Byte
Status: Read/Write. Chip: CIA-B

$BFD500 TAHI

Timer A High Byte
Status: Read/Write. Chip: CIA-B

$BFD600 TBLO

Timer B Low Byte
Status: Read/Write. Chip: CIA-B

$BFD700 TBHI

Timer B High Byte
Status: Read/Write. Chip: CIA-B

Location Range: $BFD800-$BFDB00

CIA-B TOD Counters

This 24-bit timer serves as a horizontal blank counter for the graphics.library, and is used to synchronize the output of graphics with the position of the video beam.

The CIA-B chip's TICK line is connected to the Agnus chip's _HSYNC line, so the TOD clock's counter is incremented once every horizontal blank (approximately 31,500 times a second).

$BFD800 TODLO

TOD Counter Low Byte
Status: Read/Write. Chip: CIA-B
This register holds the TOD clock's bits 0-7.

$BFD900 TODMID

TOD Counter Mid Byte
Status: Read/Write. Chip: CIA-B
This register holds the TOD clock's bits 8-15.

$BFDA00 TODHI

TOD Counter High Byte
Status: Read/Write. Chip: CIA-B
This register holds the TOD clock's bits 16-23.

$BFDB00 TODHR

Unused
Status: Read/Write. Chip: CIA-B
This register is not connected to the Amiga and therefore it is not used.

$BFDC00 SDR

Serial Data Register
Status: Read/Write. Chip: CIA-B
The CIA chip has an on-chip serial port, which allows you to send or receive a byte of data one bit at a time, with the most significant bit (bit 7) being sent first.

The CIA-B's SP line, which carries bits to the Serial Data Register, is connected to bit 0 of this chip's Data Port A, and, in turn, to pin 11 of the parallel port. The chip's CNT line, which carries the bits that are output by the Serial Data Register, is connected to bit 1 of this chip's Data Port A, and, in turn, to pin 12 of the parallel port.

For more information about the use of this register, see the entry for location $BFEC01. The Amiga's operating system does not use this register.

$BFDD00 ICR

Interrupt Control Register
Status: Read/Write. Chip: CIA-B

Bit 0: Read—did Timer A count down to 0? (1 = yes)
Write—enable or disable Timer A interrupt (1 = enable; 0 = disable)

Bit 1: Read—did Timer B count down to 0? (1 = yes)
Write—enable or disable Timer B interrupt (1 = enable; 0 = disable)
Bit 2: Read—did TOD clock reach alarm time? (1 = yes)
Write—enable or disable TOD clock alarm interrupt (1 = enable; 0 = disable)
Bit 3: Read—did the Serial Data Register finish a byte? (1 = yes)
Write—enable or disable Serial Data Register interrupt (1 = enable; 0 = disable)
Bit 4: Read—was a signal sent on the FLAG line? (1 = yes)
Write—enable or disable FLAG line interrupt (1 = enable; 0 = disable)
Bit 5: Unused. Always returns a 0 when read
Bit 6: Unused. Always returns a 0 when read
Bit 7: Read—did any CIA-A source cause an interrupt? (1 = yes)
Write—set or clear bits of this register (1 = bits written with 1 will be set; 0 = bits written with 1 will be cleared)

This register is used to control the five interrupt sources on the 8520 CIA chip. For details on its operation, see the entry for $BFED01.

The main difference between the two CIA chips pertaining to this register is that on CIA-B, the FLAG line is connected to the disk controller's DSKINDEX line. And in order for the Amiga to recognize a CIA-B chip interrupt, bit 13 of the INTENA register

Must be set to 1. See registers INTENA and INTENAR at $DFF09A and $DFF01C for more information on Amiga interrupts.

Location Range: $BFDE00-$BFDF00

CIA-B Control Registers A and B
See locations $BFEE01 and $BFEF01 for details.

$BFDE00 CRA

Control Register A
Status: Read/Write. Chip: CIA-B

Bit 0: START: Start Timer A (1 = start; 0 = stop)
Bit 1: PBON: Select Timer A output on Data Port B (1 = Timer A output appears on bit 6 for Data Port B)
Bit 2: OUTMODE: Data Port B output mode (1 = toggle bit 6; 0 = pulse bit 6 for ten microprocessor cycles)

Bit 3: RUNMODE: Timer A run mode (1 = one-shot; 0 = continuous)
Bit 4: LOAD: Force latched value to be loaded to Timer A counter (1 = force load strobe)
Bit 5: INMODE: Timer A input mode (1 = count signals on CNT line at pin 1 of keyboard port); 0 = count every ten microprocessor cycles)
Bit 6: SPMODE: Serial port ($BFEC01) mode (1 = output; 0 = input)
Bit 7: Unused

$BFDF00 CRB

Control Register B
Status: Read/Write. Chip: CIA-B

Bit 0: START: Start Timer B
(1 = start; 0 = stop)
Bit 1: PBON: Select Timer B output on Data Port B
(1 = Timer B output appears on bit 7 for Data Port B)
Bit 2: OUTMODE: Data Port B output mode
(1 = toggle bit 7; 0 = pulse bit 7 for ten microprocessor cycles)
Bit 3: RUNMODE: Timer B run mode
(1 = one-shot; 0 = continuous)
Bit 4: LOAD: Force latched value to be loaded to Timer B counter
(1 = force load strobe)
Bits 5-6: INMODE: Timer B input mode
00 = Count every ten microprocessor cycles
01 = Count signals on CNT line at pin 1 of keyboard port
10 = Count each time that Timer A counts down to 0
11 = Count Timer A 0s when CNT pulses are present
Bit 7: ALARM: Select TOD write status
(1 = writing to TOD registers sets counter; 0 = writing to TOD registers sets alarm)

Location Range: $BFE001-$BFEF01

CIA-A

Locations $BFE001-$BFEF01 are used to communicate with the first of two CIA (Complex Interface Adapter) chips. However, not every location in this range is used; the 8520 CIA chip has only 16 registers, and they are spaced at 256-byte intervals. The addresses found between these locations are mirror images of the chip's registers and should not be read from or written to. You should always use the official register

addresses given here. If you do not, you run the risk of your program not working on future model Amigas.

The CIA chips provide several functions: two eight-bit I/O ports, two multipurpose timers, one special event (TOD) timer, a serial shift register, and the ability to generate interrupts.

The 8520 CIA chip has quite a family tree. It's the grandchild of the PIA and VIA chips found in Commodore's PET and VIC-20 computers, and the direct descendant of the Commodore 64's 6526 CIA chips. In fact, except for four registers, the 6526 and 8520 are identical in operation. The difference is found in their TOD (Time Of Day) clocks: The 6526 has a TOD clock that keeps time in hours:minutes:seconds:tenths-of-seconds format. The 8520, on the other hand, replaces this clock with a 24-bit counter that counts up from 0 to 16,777,215. The term TOD has simply been passed down to the 8520; it really doesn't describe the counter at all.

The CIA chip found here is referred to as CIA-A. It serves several functions: It acts as a timer for the operating system, controls the fire button pins on both game ports, provides a status register for the disk controller, toggles the power LED on and off (which also toggles the audio cutoff filter on and off on all Amigas other than the 1000), reads the keyboard, and handles all parallel port I/O. The CIA-A also has an interrupt line that's connected to the Paula chip, so interrupts can be triggered by the chip's timers counting down, by the chip's serial port (which is connected to the keyboard), or by the parallel port's ACK line going low.

Unlike the custom chip registers, the CIA's registers are all one byte long, and each can be both read and written to.

Location Range: $BFE001-$BFE301

CIA-A Data Ports A and B

These registers allow the Amiga to communicate with the outside world. Bits of data written to the PRA and PRB registers ($BFE001 and $BFE101) can be sent to external devices, while bits of data that those devices send can be read here.

A bit can be set up for only input or output, not both at once. To define the I/O status of a bit, you use the Data Direction registers. Register DDRA ($BFE201) controls Data Port A and register DDRB ($BFE301) controls Data Port B. To set up bit 0 in data port A for output, set bit 0 in DDRA equal to 1. Now, anything you write to PRA bit 0 is output on that line. Setting DDRA bit 0 equal to 0 allows you to read the status of that line by reading PRA's bit 0.

The DDRA register is preset for normal operation when the Amiga is first turned on. In general, you should leave this register alone. In fact, changing some of Data Port A's lines from output to input, or vice versa, not only doesn't make sense, it can cause

your computer to lock up (no harm will come to your computer; you will simply be forced to reboot). For example, setting up the Amiga's power LED (which is connected to PRA's bit 1) for input would be ridiculous—what type of information could the LED provide? And setting up bit 0 for input will always lock up your computer. About the only exceptions to this rule are the fire button lines on the game controller ports (PRA bits 6 and 7). You can read these lines to determine whether a joystick's fire button or the left mouse button is being pressed, and you can also output digital data on these lines.

Bits 6 and 7 of Data Port A can also be used as an output by either Timer A or Timer B of this chip. It is possible to set a mode in which the timers do not cause an interrupt when they run down (see the description of the control registers at $BFED01, $BFEE01, and $BFEF01). Instead, they cause the output on bit 6 or 7 of Data Port B (PRB) to change. Timer A can be set either to pulse the output of bit 6, or to toggle that bit from 1 to 0 to 1. Timer B can use bit 7 of this register for the same purpose.

$BFE001 PRA

Peripheral Data Register for Data Port A
Status: Read/Write. Chip: CIA-A

Bit 0: OVL: Memory overlay bit (Always set to 0. Don't change!)
Bit 1: LED: Power LED/cutoff filter (1 = Power LED dimmed and cutoff filter inactive; 0 = Power LED at full brightness and cutoff filter active)
Bit 2: CHNG: Disk change (1 = no change; 0 = disk has been removed)
Bit 3: WPRO: Disk write protect (1 = unprotected; 0 = protected)
Bit 4: TK0: Disk track 0 (1 = head not on track 0; 0 = head on track 0)
Bit 5: RDY: Disk ready (1 = not ready for commands; 0 = ready for commands)
Bit 6: FIR0: Fire button on port 1, pin 6 (0 = button pressed)
Bit 7: FIR1: Fire button on port 2, pin 6 (0 = button pressed)

The PRA register serves a variety of functions. Bit 1 toggles the brightness of the Amiga's power LED. This bit also affects the on-off status of the audio cutoff filter, except on the Amiga 1000 and older 2000s that don't have composite mono video output. (On these model Amigas, the cutoff filter cannot be disabled through software.) Normally, the LED is at full intensity and the cutoff filter is active. Setting this bit to 1, however, dims the LED and turns off the cutoff filter.

The audio cutoff filter is a device that prevents *antialiasing*—high frequency noise caused by the sound digitization process. When on, the filter blocks most frequencies above 7KHz. When off, the Amiga may output frequencies as high as 15KHz. This creates brighter sounds, as well as some high-frequency distortion.

Bits 2 through 5 are connected to the Amiga's disk controller. These bits return the status of the currently active disk drive. The CIA-B's PRB register ($BFD100) allows you to select which drive is active. These bits will tell you if a disk has been removed from the active drive; if the disk in the active drive is write protected; if the drive head of the active drive is over track 0; and if the active drive's motor is on, running at full speed, and ready to receive a read or write command.

The last two bits, bits 6 and 7, allow you to read the left mouse button or the fire button on a joystick. Bit 6 reads the mouse/fire button for devices plugged into port 1, and bit 7 reads the mouse/fire button for devices plugged into port 2. Using these bits you can also output data through pin 6 on either game port: Simply set bit 6 or 7 (depending on which port you wish to output data on) in the DDRA register to 1 for output and then toggle the same bit in the PRA register. Setting the PRA bit equal to 1 outputs approximately +5 volts on that line. Setting the PRA bit equal to 0 pulls the line low to 0 volts. It is common courtesy to set the data direction registers back to their original values after using them.

$BFE101 PRB

Peripheral Data Register for Data Port B
Status: Read/Write. Chip: CIA-A

Bit 0: D0: Parallel port pin 2
Bit 1: D1: Parallel port pin 3
Bit 2: D2: Parallel port pin 4
Bit 3: D3: Parallel port pin 5
Bit 4: D4: Parallel port pin 6
Bit 5: D5: Parallel port pin 7
Bit 6: D6: Parallel port pin 8
Bit 7: D7: Parallel port pin 9

This register is used exclusively by the Amiga's parallel port. It controls the port's data lines, and is therefore responsible for all parallel port transmissions. For example, whenever the Amiga sends a character to a printer that's connected to the parallel port, it uses this register. It just sets all the bits in DDRB equal to 1 and writes the byte of data to be output here.

$BFE201 DDRA

Data Direction Register A
Status: Read/Write. Chip: CIA-A

Bit 0: Select bit 0 of PRA for input or output.
Must be set to 1 (output)

Bit 1: Select bit 1 of PRA for input or output.
Must be set to 1 (output)
Bit 2: Select bit 2 of PRA for input or output.
Must be set to 0 (input)
Bit 3: Select bit 3 of PRA for input or output.
Must be set to 0 (input)
Bit 4: Select bit 4 of PRA for input or output.
Must be set to 0 (input)
Bit 5: Select bit 5 of PRA for input or output.
Must be set to 0 (input)
Bit 6: Select bit 6 of PRA for input or output.
Normally set to 0 (0 = input; 1 = output)
Bit 7: Select bit 7 of PRA for input or output.
Normally set to 0 (0 = input; 1 = output)

$BFE301 DDRB

Data Direction Register B
Status: Read/Write. Chip: CIA-A

Bit 0: Select bit 0 of PRB for input or output.
(0 = input; 1 = output)
Bit 1: Select bit 1 of PRB for input or output.
(0 = input; 1 = output)
Bit 2: Select bit 2 of PRB for input or output.
(0 = input; 1 = output)
Bit 3: Select bit 3 of PRB for input or output.
(0 = input; 1 = output)
Bit 4: Select bit 4 of PRB for input or output.
(0 = input; 1 = output)
Bit 5: Select bit 5 of PRB for input or output.
(0 = input; 1 = output)
Bit 6: Select bit 6 of PRB for input or output.
(0 = input; 1 = output)
Bit 7: Select bit 7 of PRB for input or output.
(0 = input; 1 = output)

Location Range: $BFE401-$BFE701

CIA-A Timers A and B Low and High Bytes

These four timer registers (two for each timer) have different functions depending on whether you are reading from them or writing to them. When you read from these registers, you get the present value of the Timer Counter (which counts down from its initial value to 0). When you write to these registers, the value is stored in the Timer Latch, and from there it can be used to load the Timer Counter using the LOAD bit of the CRA or CRB registers (see locations $BFEE01 and $BFEF01).

These interval timers can hold a 16-bit number from 0 to 65,535, in low-byte, high-byte format (Value = LowByte + 256 * HighByte). Once the Timer Counter is set to an initial value, and the timer is started, the timer will decrement every ten microprocessor cycles. Since the clock speed of the Amiga (using the American NTSC television standard) is 7,159,090 cycles per second, every count takes approximately a millionth of a second (10 / 7,159,090 = 0.0000013968256). The formula for calculating the amount of time it will take for the timer to count down from its latch value to 0 is:

Time = LatchValue / CountSpeed

where **LatchValue** is the value written to the low and high timer registers (LatchValue = TimerLow + 256 * TimerHigh), and **CountSpeed** is 715,909 cycles per second for American (NTSC) standard television monitors, or 709,379 for European (PAL) monitors.

When Timer A or B reaches 0, it will set bit 0 or 1 in the Interrupt Control Register at $BFED01. If the timer interrupt has been enabled, an interrupt will take place, and the high bit of the Interrupt Control Register will be set to 1. Alternately, if the PBON bit is set, the timer will write data to bit 6 or 7 on Port B. After the timer gets to 0, it will reload the Timer Latch value, and either stop or count down again, depending on whether it is in one-shot or continuous mode (determined by bit 3 of the Control Register).

Although usually a timer will be used to count every ten microprocessor cycles, Timer A can also count external pulses on the CNT line, which is connected to the Amiga's keyboard.

Timer B is even more versatile. In addition to these two sources, Timer B can count the number of times that Timer A goes to 0. By setting Timer A to count the microprocessor clock, and setting Timer B to count the number of times that Timer A reaches 0, you effectively link the two timers into one 32-bit timer that can count for more than an hour before resetting.

In the Amiga, CIA-A Timer A is used to time the transfer of keyboard data. Timer B is used by Exec (the core of the Amiga's multitasking operating system) to synchronize task switching; it also serves as the timer.device's UNIT_MICROHZ timer. As a result, neither Timer A or B is available for general use.

$BFE401 TALO
Timer A Low Byte
Status: Read/Write. Chip: CIA-A

$BFE501 TAHI
Timer A High Byte
Status: Read/Write. Chip: CIA-A

$BFE601 TBLO
Timer B Low Byte
Status: Read/Write. Chip: CIA-A

$BFE701 TBHI
Timer B High Byte
Status: Read/Write. Chip: CIA-A

Location Range: $BFE801-$BFEB01
CIA-A TOD Counters

In addition to the two general-purpose timers, the 8520 CIA chip has a special-purpose 24-bit counter that can count to 16,777,215 ($FFFFFF). Unlike the chip's other two timers, the TOD clock counts up, and when the TOD clock reaches is maximum value, it starts over at 0 and continues counting. The TOD clock can also be set to generate an interrupt whenever it reaches a determined value.

The TOD clock increments its counter approximately 60 times a second (50 times a second on European PAL systems)—the same frequency as the AC voltage that powers your computer. In fact, on the Amiga 1000, the CIA-A chip's TICK line (which is used to drive the TOD clock) is connected directly to the computer's power line tick. On the Amiga 500, this line is connected to the Agnus chip's _VSYNC line, which changes state during each vertical blank (again, about 60 times a second). By default, the Amiga 2000 uses the power line tick, as does the 1000. But by changing the 2000's J300 jumper, you can force it to conform to Amiga 500 standards and count vertical blanks instead.

It takes three eight-bit registers to hold the TOD clock's 24-bit timer value: a low-byte register, a mid-byte register, and a high-byte register. The actual timer value can be determined with the formula HighByte * 65536 + MidByte * 256 + LowByte.

The TOD registers can be used for two purposes, depending on whether you are reading them or writing to them. If you are reading them, you will always be reading the timer's current value. There is a latching feature associated with reading the high-byte register ($BFEA01) in order to solve the problem of the counter changing while you are reading the registers. For example, if you were reading the high-byte register just as the counter was changing from $01FFFF to $020000, it is possible that you would read the $01 in the high-byte register, and by the time you read the mid-byte register it would have changed from $FF to $00. Therefore, you would read $010000 instead of either $01FFFF or $020000.

To prevent this kind of mistake, the TOD registers stop updating as soon as you read the high-byte register, and do not start again until you read the low-byte register. Of course, the clock continues to keep time internally even though it does not update the registers. If you want to read the mid-byte or the low-byte, there is no problem, and no latching will occur. But anytime you read the high-byte, you must follow it by reading the low-byte, even if you don't care about its value, or else the registers will not continue to update.

Writing to these registers either sets the counter or the alarm time, depending on the setting of bit 7 of Control Register B (CRB, $BFEF01). If that bit is set to 1, writing to the TOD registers sets the alarm. If the bit is set to 0, writing to the TOD registers sets the TOD clock. In either case, as with reading the registers, there is a latch function. This function stops the clock from updating when you write to the high-byte register. The clock will not start again until you write to the low-byte register.

The CIA-A's TOD clock serves as the timer.device's UNIT_VBLANK timer. Unlike the Timer A-driven UNIT_MICROHZ clock (which is also used by Exec), UNIT_VBLANK is not slowed down if the computer is running several programs at once.

$BFE801 TODLO

TOD Counter Low Byte
Status: Read/Write. Chip: CIA-A
This register holds the TOD clock's bits 0-7.

$BFE901 TODMID

TOD Counter Mid Byte
Status: Read/Write. Chip: CIA-A

This register holds the TOD clock's bits 8-15.

$BFEA01 TODHI
TOD Counter High Byte
Status: Read/Write. Chip: CIA-A
This register holds the TOD clock's bits 16-23.

$BFEB01 TODHR
Unused
Status: Read/Write. Chip: CIA-A
This register is not connected to the Amiga and therefore it is not used.

$BFEC01 SDR
Serial Data Register
Status: Read/Write. Chip: CIA-A
The CIA chip has a built-in serial port, which allows you to send or receive a byte of data one bit at a time, with the most significant bit (bit 7) being transferred first. Control Register A at $BFEE01 allows you to choose input or output modes. In input mode, a bit of data is read from the chip's SP line whenever a signal on the CNT line appears. After eight bits are received this way, the data is placed in the Serial Data Register, and an interrupt is generated to let you know that the register should be read.

In output mode, you write data to the Serial Data Register, and it is sent out over the SP line, using Timer A for the baud rate generator. Whenever a byte of data is written to this register, transmission will start as long as Timer A is running in continuous mode. Data is sent at half the Timer A rate, and an output will appear on the CNT line whenever a bit is sent. After all eight bits have been sent, an interrupt is generated to indicate that it is time to load the next byte to send into the Serial Data Register.

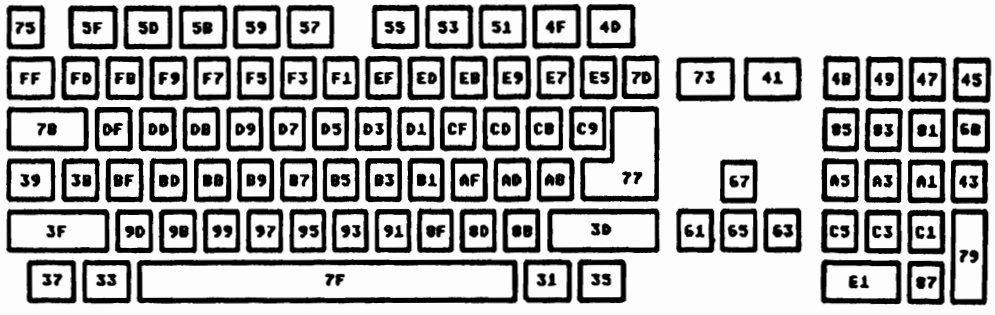
The Amiga uses this register to receive data from the keyboard. The CIA-A chip's SP line is connected to the keyboard's KDAT line (via pin 2 of the keyboard port), and the CNT line is connected to the keyboard's KCLK line (via pin 1 of the keyboard port).

By reading this register directly, you can find the *rawkey* value of the current key being pressed. A rawkey value specifies the physical location of a key on the keyboard. Normally, the Amiga's operating system reads this value from the Serial Data Register and uses the computer's current keymap table (such as USA1) to convert it to ASCII. Keymap tables are important because different countries use slightly different keyboard layouts. For example, the rawkey value for a *Z* on an American keyboard is the same as the rawkey value for a *Y* on a German keyboard. There are at least 15 different

Amiga keyboard configurations in existance.

Figure 3-5 shows how the rawkey values relate to the Amiga's keyboard layout.

Figure 3-5: Amiga Rawkey Codes



The numbers shown in Figure 3-5 reflects the hex value returned when a key is pressed. The keyboard also sends the computer rawkey codes when keys are released. A key's release value is equal to its key down value plus 128 (its high bit is set).

You should know that rawkey codes don't change if the shift key is being pressed. Instead, the shift key has been given a rawkey code of its own, and the Amiga reads these codes to keep track of the shift key's up/down status. The same is true with the other special keys—Control, Alt, and both the left and right Amiga keys. Interestingly, the Caps Lock key doesn't return a rawkey code when it's released; it sends a key down code when it's first pressed (the Caps Lock LED lights up), and a key release code when it's pressed again (the Caps Lock LED goes out).

There are six additional codes that can be sent to the Serial Data Register. These codes announce special conditions that can occur. If Control-Amiga- Amiga is pressed (the computer is about to be rebooted), the keyboard sends a $78; if the last rawkey code sent was garbled because of a transmission error, the keyboard sends an $F9 and then attempts to transmit the errant rawkey code again; if the keyboard's internal 10-character keyboard buffer is full, the keyboard sends an $FA; if the keyboard's initial self-test fails (this is performed when the computer is first turned on), the keyboard sends an $FC and blinks its Caps Lock light off and on to give a visual warning to the user; and if any keys were being pressed when the computer was first turned on, the byte

$FD is sent, followed by the key codes of all the keys that were pressed, followed by a terminating byte of $FE.

$BFED01 ICR

Interrupt Control Register
Status: Read/Write. Chip: CIA-A

Bit 0: Read—did Timer A count down to 0? (1 = yes)
Write—enable or disable Timer A interrupt (1 = enable; 0 = disable)
Bit 1: Read—did Timer B count down to 0? (1 = yes)
Write—enable or disable Timer B interrupt (1 = enable; 0 = disable)
Bit 2: Read—did TOD clock reach alarm time? (1 = yes)
Write—enable or disable TOD clock alarm interrupt (1 = enable; 0 = disable)
Bit 3: Read—did the Serial Data Register finish a byte? (1 = yes)
Write—enable or disable Serial Data Register interrupt (1 = enable; 0 = disable)
Bit 4: Read—was a signal sent on the FLAG line? (1 = yes)
Write—enable or disable FLAG line interrupt (1 = enable; 0 = disable)
Bit 5: Unused. Always returns a 0 when read
Bit 6: Unused. Always returns a 0 when read
Bit 7: Read—did any CIA-A source cause an interrupt? (1 = yes)
Write—set or clear bits of this register (1 = bits written with 1 will be set; 0 = bits written with 1 will be cleared)

This register is used to control the five interrupt sources on the 8520 CIA chip. These sources are Timer A, Timer B, the TOD clock, the Serial Data Register, and the chip's FLAG line. Timers A and B cause an interrupt when they count down to 0. The TOD clock generates an interrupt when it reaches the alarm time. The Serial Data Register interrupts when it compiles eight bits of input or output. An external signal pulling the CIA chip's FLAG line low will also cause an interrupt (on CIA-A, the FLAG line is connected to the ACK line—pin 10—of the parallel port).

Even if the conditions for a particular interrupt is satisfied, the interrupt must still be enabled for an interrupt to occur. This is done by writing to the Interrupt Control Register. What happens when you write to this register depends on the way that you set bit 7. If it is set to 0, any other bit that is written to with a 1 will be cleared, and the corresponding interrupt will be disabled. If you set bit 7 to 1, any bit written to with a 1 will be set, and the corresponding interrupt will be enabled. In either case, the interrupt enable flags for those bits written to with a 0 will not be affected.

For example, to disable all interrupts from machine language, you could MOVE.B #$7F,$BFED01. This sets bit 7 to 0, which clears all of the other bits, since they are all written to with 1s. Don't try this while multitasking is still turned on, as it will turn off Timer B, which handles task switching.

To turn on Timer B interrupt, a program could MOVE.B #$82,$BFED01. Bit 7 is set to 1 and so is bit 1, so the interrupt which corresponds to bit 1 (Timer B) is enabled.

When you read this register, you can tell if any of the conditions for a CIA interrupt were satisfied because the corresponding bit will be set to a 1. For example, if Timer A counts down to 0, bit 0 of this register will be set to 1. If in addition, the enable bit that corresponds to that interrupt source is set to 1, and an interrupts occurs, bit 7 will also be set. This allows a multi-interrupt system to read one bit and see if the source of a particular interrupt is CIA-A. You should note, however, that reading this register clears it, so you should preserve its contents if you want to test more than one bit. In order for the Amiga to recognize a CIA-A chip interrupt, bit 3 of the INTENA register must be set to 1. See registers INTENA and INTENAR ($DFF09A and $DFF01C) for more information.

$BFEE01 CRA

Control Register A
Status: Read/Write. Chip: CIA-A

Bit 0: START: Start Timer A
(1 = start; 0 = stop)
Bit 1: PBON: Select Timer A output on Data Port B
(1 = Timer A output appears on bit 6 for Data Port B)
Bit 2: OUTMODE: Data Port B output mode
(1 = toggle bit 6; 0 = pulse bit 6 for ten microprocessor cycles)
Bit 3: RUNMODE: Timer A run mode
(1 = one-shot; 0 = continuous)
Bit 4: LOAD: Force latched value to be loaded to Timer A counter
(1 = force load strobe)
Bit 5: INMODE: Timer A input mode
(1 = count signals on CNT line at pin 1 of keyboard port);
0 = count every ten microprocessor cycles)
Bit 6: SPMODE: Serial port ($BFEC01) mode
(1 = output; 0 = input)
Bit 7: Unused

Bits 0-3. These four bits control Timer A. Bit 0 is set to start the timer counting down, and set to 0 to stop it. Bit 3 sets the timer for one-shot or continuous mode. In one-shot mode, the timer counts down to 0, sets the counter value back to the latch value, and then sets bit 0 back to 0 to stop the timer. In continuous mode, it reloads the latch value and starts all over again.

Bits 1 and 2 allow you to send a signal on bit 6 of Data Port B when the timer counts. Setting bit 1 to 1 forces this output (which overrides DDRB's bit 6, and the normal Data Port B value). Bit 2 allows you to choose the form this output to bit 6 of Data Port B will take. Setting bit 2 to a value of 1 will cause bit 6 to change value when the timer runs down (a value of 1 will change to 0, and a value of 0 will change to 1). Setting bit 2 to a value of 0 will cause a single pulse of a ten-microprocessor-cycle duration (about a millionth of a second) to occur.

Bit 4. This bit is used to load the Timer A counter with the value that was previously written to the Timer low- and high-byte registers. Writing a 1 to this bit will force the load (although there is no data stored here, and the bit has no significance when read).

Bit 5. Bit 5 is used to control just what it is Timer A is counting. If this bit is set to 1, the timer counts pulses on the CNT line, which is connected to pin 1 on the keyboard port. If the bit is set to 0, it counts every ten microprocessor cycles (approximately once every millionth of a second). The Amiga sets this bit equal to 1 so the keyboard can drive Timer A and control the transfer rate of data that is sent to the Serial Data Register.

Bit 6. Whether the Serial Data Register ($BFEC01) is currently inputting or outputting data is controlled by this bit. This bit is always set to 0 (for input mode) so the Amiga can read incoming keyboard data.

$BFEF01 CRB

Control Register B
Status: Read/Write. Chip: CIA-A

Bit 0: START: Start Timer B (1 = start; 0 = stop)
Bit 1: PBON: Select Timer B output on Data Port B
(1 = Timer B output appears on bit 7 for Data Port B)
Bit 2: OUTMODE: Data Port B output mode
(1 = toggle bit 7; 0 = pulse bit 7 for ten microprocessor cycles)
Bit 3: RUNMODE: Timer B run mode (1 = one-shot; 0 = continuous)
Bit 4: LOAD: Force latched value to be loaded to Timer B counter
(1 = force load strobe)

Bits 5-6: INMODE: Timer B input mode
00 = Count every ten microprocessor cycles
01 = Count signals on CNT line at pin 1 of keyboard port
10 = Count each time that Timer A counts down to 0
11 = Count Timer A 0s when CNT pulses are present

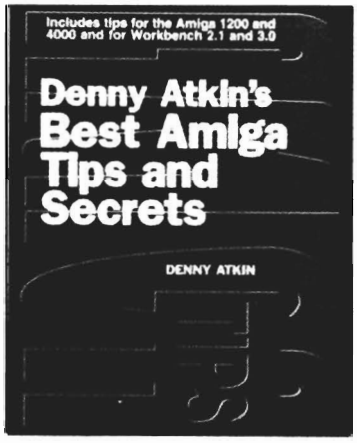
Bit 7: ALARM: Select TOD write status (1 = writing to TOD registers sets counter; 0 = writing to TOD registers sets alarm)

Bits 0-3. These four bits perform the same functions for Timer B that bits 0-3 of Control Register A perform for Timer A, except that Timer B output on Data Port B appears at bit 7, and not bit 6.

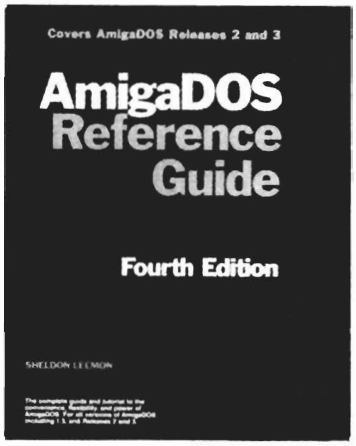
Bits 5 and 6. These two bits are used to select what Timer B counts. If both bits are set to 0, Timer B counts every ten microprocessor cycles (which occur at the rate of approximately 715,909 cycles per second). If bit 6 is set to 0 and bit 5 is set to 1, Timer B counts Timer A underflow pulses, which is to say that it counts the number of times that Timer A counts down to 0. This is used to link the two timers into one 32-bit timer that can count for over an hour without resetting. Finally, if both bits are set to 1, Timer B counts the number of times that Timer A counts down to 0 *and* there is a signal on the CNT line (pin 1 of the keyboard port).

Bit 7. This bit controls what happens when you write to the TOD registers. If this bit is set to 1, writing to the TOD registers sets the alarm time. If this bit is set to 0, writing to the TOD registers sets the TOD clock.

COMPUTE Books Is Your Amiga Resource



Denny Atkin's Best Amiga Tips and Secrets



AmigaDOS Reference Guide Fourth Edition

_____ Denny Atkin's Best Amiga Tips and Secrets $19.95 (2753)

_____ AmigaDOS Reference Guide, Fourth Edition $22.95 (2680)

_____ **Total**

_____ Tax (Residents of NC, NJ, and NY add appropriate sales tax.)

_____ Shipping and handling (U.S.: $2.50; Canada: $4.00; Foreign: $6.00)

_____ **Grand Total**

____ **Check or Money Order** ____ **MC** ____ **VISA**

Signature _____
(Required)

Acct. No. _____ Exp. Date _____

Name _____

Street Address _____

City _____ State _____ ZIP _____

Mail to:
COMPUTE Books
c/o CCC
2500 McClellan Ave.
Pennsauken, NJ 08109

Please allow four to six weeks for delivery. Offer good while supplies last. All orders must be paid in U.S. funds drawn on a U.S. bank.

Hardware Index

FasTrig

Four trigonometric functions are used in almost all graphic, video, audio, simulation, and engineering computer programs. The Sine, Cosine, Tangent, and Arctangent probably are critical in the code YOU write. And if they are, they also are a critical bottleneck to your execution speed. ANSI Standard IEEE floating point trig functions can be used at a rate of only 200 calls per second on a conventional 7.5 MHz Amiga®.

What to do? Live with slow programs? Buy an accelerator? Buy an FPP? What to tell your clients to do? Should they be willing to buy programs that execute slowly? Should they accept programs that require substantial hardware investment to run at reasonable speeds? If your clients don't accept these "solutions" gracefully, why do you?

You are no longer forced to use slow trigonometry functions!

FasTrig provides these same critical functions in object modules that execute at 36,000 calls per second on conventional 7.5 MHz Amigas! No special hardware! No accelerators! No floating point processors! Yes - you do have to change software calls.

Object modules work on any Amiga and with any of the floating point libraries you may be using. (In fact, they do not use floating point arithmetic). *FasTrig* achieves its speed by treating angles and trig functions with integer binary arithmetic. *FasTrig* lets your Amiga fly by doing only the kinds of operations that computers do quickest.

Object modules for Sine, Cosine, Tangent, and Arctangent functions can be linked into your own executable tasks. Support modules allow you to interface with conventional floating point arithmetic if desired. Source codes are provided in C and Assembly languages. Example codes demonstrate use of the functions. A sixty page document manual fully discusses the *FasTrig* concept, and a twenty-five page implementation manual details the *FasTrig* diskette contents and their use on the Amiga.

FasTrig
For All Amiga Operating Systems
One Diskette - Not Copy Protected
Document Manual
Implementation Manual
$55.00

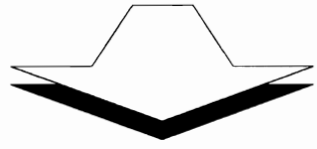
Curious - But Not Committed?
The *FasTrig* Document Manual:
"Fast Trigonometry Using
Binary Integer Arithmetic"
Can Be Purchased Alone for $5.50

Parth Galen Software
Post Office Box 482
Cold Spring MN 56320
(612) 685-8871

Argonauts

Beta

Commercial

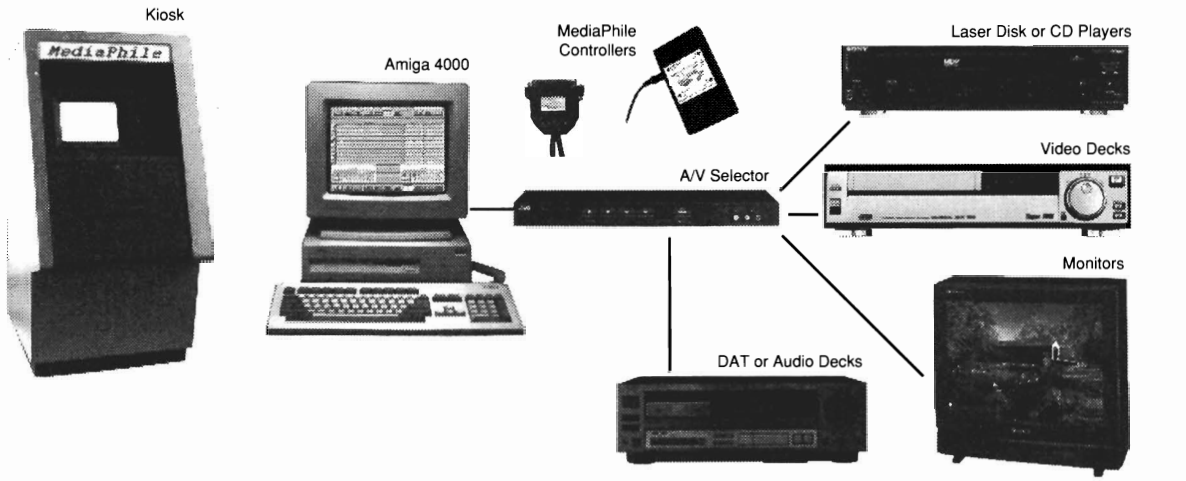


Caught your attention didn't I. **Rule #1** in advertising, if the customer doesn't read the ad no matter how good the product it won't sell. **Rule #2** get to the point fast. Argonauts is a quarterly newsletter for the commercially active Amiga entrepreneur or developer. If you're reading this then you have the technical talent. Argonauts is about what happens next. Product ideas, first person business accounts, news from around the world in different Amiga markets. A talent registry so you can let others know what you can do along with who and where you are. Cash challenge prizes, interesting research papers a developer's roundtable and more.
Rule #3 Tell them how much and where to send it:

$24.95 in US $30 everywhere else.
Overseas includes airmail delivery
US funds drawn on US bank only.
Make check or money order payable to: Morning Star International.

Send to:
Argonauts Dept. CO
Box 94
Pearl River NY 10965
USA

Multimedia Control



MediaPhile® edit controllers and executive library programs give complete interactive control over video decks and camcorders from your software. MediaPhile controllers are being used in video post production, training, multimedia presentation, and in point-of-sale and information kiosks.

Multitasking control is provided from VISCA, VBOX, Video Director, Nucleus, Future Video, GVP and MediaPhile controllers. Send ARexx messages or make C-program calls to read tape position and control your devices. Control any industrial, prosumer or consumer video deck or camcorder. Control infrared devices like compact disk players, and audio decks with MediaPhile infrared controllers. Arrange for custom serial port control of any device with two week turnaround.

Our staff will work with you to develop practical working solutions to your control problems at costs your customers can afford. Programming and electronic design services are available for your custom control requirements. Give us a call to discuss your needs.

Complete Systems
Call For Quote

Control From
$450.

Interactive MicroSystems, Inc.
9 Red Roof Lane, Salem NH 03079

Telephone: (603) 898-3545
Facsimile: (603) 898-3606

THE ULTIMATE AMIGA REFERENCE

COMPUTE's *Mapping the Amiga* is more than just another programming book for the Amiga. Nowhere else will you find a complete alphabetical listing of library functions — including new Revision 2.0 and 3.0 functions — with syntax given in both C and machine language. This indispensable reference guide is the only source that includes full descriptions of every Amiga hardware register and an element-by-element breakdown of each Amiga system structure.

Inside *Mapping the Amiga* you'll find:

- **A complete listing of the Amiga function calls**
- **Descriptions of all system structures**
- **Detailed explanations of the computer's hardware — with every register documented**
- **New Amiga OS revision 2.0 and 3.0 programming information, including descriptions of AGA-specific function calls**

Whether you're an expert programmer or a beginner, *Mapping the Amiga* is the book you'll want next to your computer.

Authors Randy Thompson and Rhett Anderson are the founding editors of COMPUTE's *Amiga Resource* magazine and currently work as professional game programmers. Together they wrote the hot Amiga arcade game *Nova 9* and aided in the development of other Amiga titles, including *A-10 Tank Killer* 1.5. Thompson is also the author of *PC SpeedScript* and the popular Amiga programs *HotKey!* and *X-Ray*. Anderson is the author of the revolutionary Sliced HAM Amiga video mode.

$27.95 US
£25.95 UK

ISBN 0-87455-267-2

90000

9 780874 552676